1991-12

Application of neural network to adaptive control theory for super-augmented aircraft

Bertrand, Denis J. S. R.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

Application of Neural Network to Adaptive
Control Theory for Super-Augmented Aircraft

by

Denis J.S.R. Bertrand

December 1991

Thesis Advisor: Dan J. Collins

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a REPORT SECURITY CLASSIFICATION: Unclassified	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION: Naval Postgraduate School; 6b OFFICE SYMBOL (If applicable)	7a NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
6c ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000	7b ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000
8a NAME OF FUNDING/SPONSORING ORGANIZATION; 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO; TASK NO; WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
Application of Neural Network to Adaptive Control Theory for Super-Augmented Aircraft

12 PERSONAL AUTHOR(S)
Bertrand, Denis J.S.R.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year Month Day)	15 PAGE COUNT
Engineer's Thesis	FROM TO	December 1991	205

16 SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy of position of the Department of Defense or the U.S. Government.

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Neural Networks, Adaptive Control, Backpropagation, Parallel Distributed Processing,

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The neural network structures developed in this thesis demonstrate the ability of parallel distributed processing in solving adaptive control problems. Adaptive control theory implies a combination of a control method and a model estimation. The control method investigated is the Lyapunov Model Reference Adaptive Control or MRAC and the model estimation investigated is the linear least square estimator. The neural network theory is introduced with emphasis on the back-propagation algorithm. The implementation of the neural network adaptive control structure is demonstrated on the longitudinal dynamics of the X-29 fighter aircraft. Three configurations are proposed to train the neural network

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION	
X UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	Unclassified	
22a NAME OF RESPONSIBLE INDIVIDUAL: Dr. D. J. Collins	22b TELEPHONE (Include Area Code): (408) 646-2311	22c OFFICE SYMBOL: 67Co

BLOCK #18 (CONT)

Super-Augmented Aircraft

BLOCK #19 (CONT)

adaptive control structures to provide the appropriate inputs to the unstable X-29 plant so that desired responses could be obtained. These configurations are presented in eight cases, which emulates stable systems like the X-29 closed-loop plant or the optimal and the limited X-29 controllers, and unstable systems like the X-29 plant or its inverse.

Approved for public release; distribution is unlimited.

Application of Neural Network to Adaptive
Control Theory for Super-Augmented Aircraft

by

Denis J.S.R. Bertrand
Captain, Canadian Army
B.E.E., Royal Military College of Kingston, Canada 1985

Submitted in partial fulfillment of
the requirements for the degrees of

MASTER OF SCIENCE IN AERONAUTICAL ENGINEERING
AND
AERONAUTICAL ENGINEER

from the

ABSTRACT

The neural network structures developed in this thesis demonstrate the ability of parallel distributed processing in solving adaptive control problems. Adaptive control theory implies a combination of a control method and a model estimation. The control method investigated is the Lyapunov Model Reference Adaptive Control or MRAC and the model estimation investigated is the linear least square estimator. The neural network theory is introduced with emphasis on the back-propagation algorithm. The implementation of the neural network adaptive control structure is demonstrated on the longitudinal dynamics of the X-29 fighter aircraft. Three configurations are proposed to train the neural network adaptive control structures to provide the appropriate inputs to the unstable X-29 plant so that desired responses could be obtained. These configurations are presented in eight cases, which emulates stable systems like the X-29 closed-loop plant or the optimal and the limited X-29 controllers, and unstable systems like the X-29 plant or its inverse.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

At this point of my thesis I would like to take the time and express my appreciation to those who contributed to the completion of this thesis. A special thank you goes to Professor Dan Collins, my thesis advisor, who suggested the topic of the thesis and provided the right amount of guidance to accomplish this research.

A thank you is also extended to Professor Louis Schmidt, who served as the second reader to this document, and to LCDR Shahar Dror, with whom I shared my laborious problems, and from whom I received valuable ideas. Dear Shahar, my wife Monika and I thankfully accept your invitation and promise you on this day, December 8, 1991, that we will visit your beautiful country Isreal in the near future.

A most sincere thank you goes to my pearl, my lovely wife Monika who I met here in Monterey, and whose constant motivation and love was a significant contribution to this thesis. Although her Austrian heritage made German her mother tongue, she conquered the English language so respectably that without her linguistic support, this French-Canadian, myself, would still find himself struggling with the English grammar.

I. INTRODUCTION

The parallel distributed processing structure of neural networks provides the models for solving adaptive control problems, as demonstrated in Ref. 1. Adaptive control involves a self-learning controller which has the ability to adjust itself in order to compensate for system changes. The control and estimation functions of adaptive control will be implemented on the H_2 and H_∞ controllers and on the plant of the X-29 [Ref. 2].

Three configurations are proposed for training the neural networks to provide the appropriate inputs to the X-29 plant in which desired responses are obtained.

This thesis will investigate the applications of methods that are based on neural network adaptive control theory to the reduced order, linearized longitudinal dynamics model of the X-29 aircraft. Chapter II introduces neural network theory and the backpropagation algorithm. In Chapter III, two traditional adaptive control methods will be briefly discussed, a one-step-ahead control algorithm will be used to design the control model, and a linear least square estimation algorithm will be implemented to design the estimator model. Chapter IV presents a description of the fighter as well as the H_2 and H_∞ controllers that were designed to solve the

subsonic longitudinal instability of that aircraft. Chapter V describes the hardware and software used for the experimental set-up, in addition to model design considerations which include design objectives, model structures selection, and configurations. Chapter VI presents the results of the three configurations' simulations. Finally, Chapter VII concludes with some remarks on what has been achieved and what is recommended for further studies.

II. NEURAL NETWORK THEORY

A. ANALOGY TO THE BRAIN

Both the brain and the digital computer operate on electrical signals, perform computational functions and are composed of a very large number of simple elements. The major difference is in the signal transmission time scale. The computer involves microsecond or even nanosecond time scales to transmit a signal compared to the slow nerve impulses. Nevertheless, the advantage of the brain is, that its huge computation rate is achieved by an enormous amount of parallel units which surpass any modern computer system.

Neural network elements are inspired by the elementary functions of the biological neuron. They are organized in such a way that they exhibit some characteristics of the human brain. That is, they have the ability to learn from experience, to perform abstractions of inputs with relevant information, and to generalize their knowledge from previous results.

By *learning* we mean that the neural network can modify its behavior from the environment's response. The neural network will self-adjust its weights to produce the desired output.

By *abstraction* we mean the ability to obtain idealized prototypes from a given set of inputs.

By *generalization* we mean that the neural network trained on input and output examples can produce a reasonable output to an input differing from those it was trained on.

Figure 1 shows the typical neuron. The neuron consists of three sections: the dendrites, the cell body, and the axon. [Ref. 3]

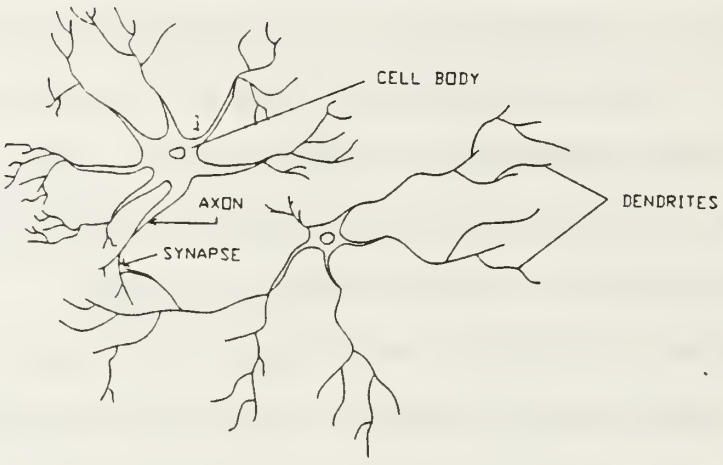


Figure 1 Biological Neuron

The dendrites receive signals from other neurons through the synapses, the connection points. These inputs are multiplied by the corresponding weight, the synapse strength, and then summed in the cell body to determine if their excitation level exceeds the specified threshold. On the affirmative, the cell fires and sends a signal to the other neurons through the axons.

B. NEURAL NETWORK ARCHITECTURE

In a neural network, we refer to the neuron as a processing element, *PE*. As shown in Fig. 2, the input paths are comparable to the dendrites, the summation operation and the transfer function to the cell body, and the output paths to the axon. [Ref. 4]

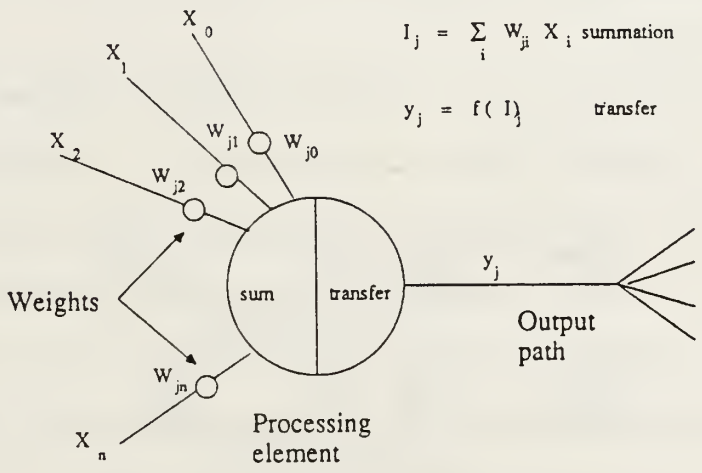


Figure 2 Building Blocks of Neural Computing Network

1. Processing Elements

A processing element is a simple computational unit which receives data from its input side, processes that data through a summation operation and a transfer function, and then sends the result to the neighboring processing elements from its output side. Figure 3 represents a simple neural network architecture. Many processing elements are grouped into layers and are either fully or randomly connected to the

processing elements of the successive layers. These connections determine in what manner the processing elements will react with each other.

The neural networks for this research consist of feedforward networks in which the connections feed the information in only one direction. No recurrent layers or feedback loops from a processing element to a previous one will be considered.

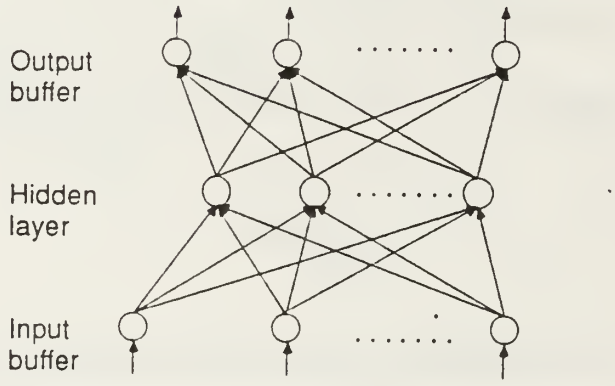


Figure 3 A Simple Neural Network Architecture

The first layer is the input buffer, which presents the information to the network. The last layer is the output buffer, which contains the response of the network to the given input. Intermediate layers are referred to as hidden layers.

Referring to control theory in state space, the input buffer elements correspond to the elements of the input variable, the output buffer elements to the ones of the output variable, and the hidden units to the elements of the state variable.

At any given time, each processing element has a certain level of activation. The pattern and the level of these activations determine the state of the system at that given point.

2. Activation function Logic

Neuralworks Professional II/ Plus® development software contains an activation function logic, shown in Fig. 4, which demonstrates the complexity of the activation function of the system. The use of complex activation functions may help the network to solve various nonlinear systems. Activations may include a summation function, a transfer function, a noise generator, scaling, limiting, thresholding, and an output function. [Ref. 4]

The transfer function could be simply a linear, or some monotonic function like the sigmoid and hyperbolic tangent functions. A nonlinear network can be achieved by using any of the last two functions since they provide nonlinear responses.

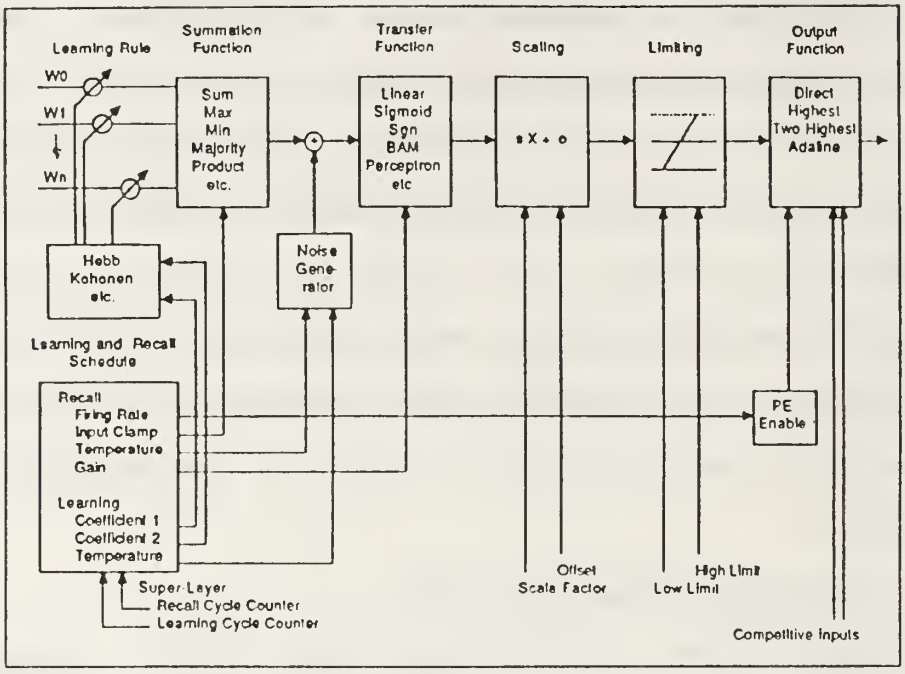


Figure 4 Neural networks Professional II Activation Function Logic

The sigmoid function, as shown in Fig. 5, is a logistic or "squashing" function. *NET* is equivalent to *x* in the following notation.

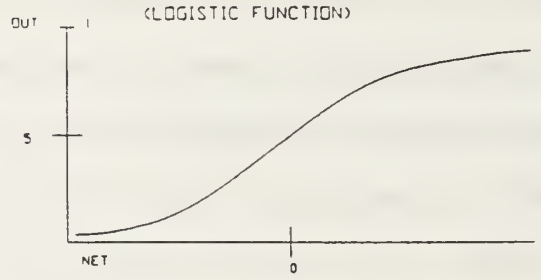


Figure 5 Sigmoidal Logistic Function

The sigmoid function is expressed as,

$$OUT = 1 / (1+e^{-x}) \tag{2.1}$$

Its derivative can be simply represented in terms of itself,

$$OUT' = e^{-x}/(1+e^{-x})^2$$

$$\text{or} \quad OUT'(x) = out(x)*(1-out(x)) \tag{2.2}$$

which saves a large amount of computation time. Equation (2.2) defines a nonlinear gain that solves the noise-saturation dilemma of Grossberg (1973); that is, it permits the network to handle both, small and large signals. The central region of high gain helps the small input signals to be processed, while the decreasing slope or gain at both, negative and positive extremes, are adequate for large signals.

Another nonlinear activation function which is often used is the hyperbolic tangent function shown in Fig. 6. It is expressed as follows:

$$OUT = tanh(x) \quad \text{or} \quad OUT = (e^{x}-e^{-x})/(e^{x}+e^{-x}) \tag{2.3}$$

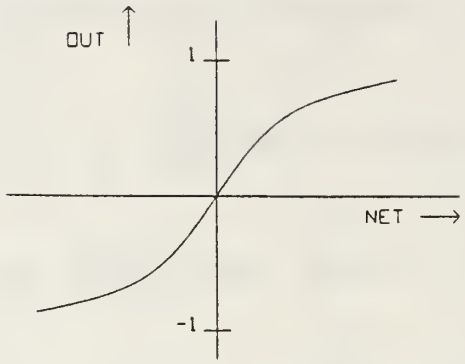


Figure 6 Hyperbolic Tangent Function

As with the sigmoid function, its derivative can also be expressed in terms of itself:

$$OUT'(x) = (1+out(x))*(1-out(x)) \qquad (2.4)$$

3. Learning rule

There are two distinct phases in the operation of the network: learning and recall. Three types of learning exist: supervised, unsupervised, and reinforcement.

Supervised learning involves modifying the connection weights in response to a desired output presented to the network corresponding to a given input. In this thesis, the generalized delta learning rule was selected to reduce the error between the actual and desired output of a processing element.

If the network is not given any desired output, an *unsupervised learning* process applies. The network forms

groups of similar input patterns, where each processing element responds strongly to different groups.

Self-supervised learning falls between the supervised and the unsupervised learning. The network will determine its own desired solution and trains itself accordingly.

4. Recall rule

Recall is a simple feedforward network where no learning takes place, i.e., no feedback between layers. An input is presented to the network, the information is propagated forward through the different layers, and an output is obtained. It is a straightforward process, where only the summation operation and the transfer function apply. It is an integral part of the testing process which compares the desired and the actual output of the network to determine the current error.

C. BACK-PROPAGATION ALGORITHM

The supervised learning rule and the back-propagation algorithm were chosen for the networks of this investigation. Back-propagation has a particular way of handling errors. It distributes the error by propagating the output error backward through the connections of the previous layers.

When the signal is fedforward from the input to the output, the error between the desired and actual output is obtained. This error is, then, multiplied by the derivative of

the transfer function, and backpropagated from the input to the output layer to adjust or update the connections' weight.

The typical neural network using the back-propagation algorithm is best represented with an input layer, an output layer, and at least one hidden layer using a monotonic activation function. [Ref. 4:pp. NC112]

For a better understanding of the next sections, a clear notation is necessary. The superscript in brackets symbolizes the layer being considered,

$X_j[s]$: current output state of jth neuron in layer *s*,

$W_{ji}[s]$: weight on connection joining ith neuron in layer *s-1* to jth neuron in layer *s*,

$I_j[s]$: weighted summation of inputs to jth neuron in layer *s*.

1. The Global Error Function

The learning process has the aim of minimizing the global error, *E*, by adjusting the weights in the network. Therefore, the measure of the global error, *E*, is achieved by subtracting the desired output *d* by the actual output *o*, as follows:

$$E = 0.5*\Sigma((d_j-o_j)^2) \qquad (2.5)$$

where the term in parentheses is the raw local error. The global error function, *E*, is also a differentiable function of

all connection weights in the network. The derivative of E with respect to I gives,

$$e_j^{[s]} = -dE/dI_j^{[s]} \tag{2.6}$$

The raw local error of equation (2.5) is scaled by multiplying it by the derivative of the transfer function, or

$$\begin{aligned} e_j &= -dE/dI_j^{[s]} \\ &= -(dE/do_j)*(do_j/dI_j) \\ &= (d_j-o_j)*f'(I_j) \end{aligned} \tag{2.7}$$

The scaled local error, which is backpropagated, will be stored in each processing element in its error field.

A gradient descent rule is used to determine if an increment or decrement of the current weights, $w_{ij}^{[s]}$, is favorable to reduce the global error. The result by using the chain rule and equation (2.6) is,

$$\begin{aligned} dE/dw_{ji}^{[s]} &= (dE/dI_j^{[s]})*(dI_j^{[s]}/dw_{ji}^{[s]}) \\ &= -e_j^{[s]}*x_i^{[s-1]} \end{aligned} \tag{2.8}$$

The gradient descent rule used is,

$$\Delta w_{ji}^{[s]} = -lcoef*(dE/dw_{ji}^{[s]}) \tag{2.9}$$

where *lcoef* is the learning coefficient or rate.

Combining equations (2.8) and (2.9) gives,

$$\Delta w_{ji}^{[s]} = lcoef * e_j^{[s]} * x_i^{[s-1]} \qquad (2.10)$$

The error signal, $e_j^{[s]}$, of the above equation is applied only to the elements of the output layer.

As to the elements of the hidden layers which do not have any desired output, $e_j^{[s]}$ is expressed as the derivative of the transfer function multiplied by the error and the weights backpropagated from the previous layer, or

$$e_j^{[s]} = f'(I_j^{[s]}) * \Sigma_k (e_k^{[s+1]} * w_{kj}^{[s+1]}) \qquad (2.11)$$

2. Back-propagation Summary

The input layer will be presented with some data which the network will propagate in a straightforward sense (feedforward) to the output layer. At the same time, all the summed inputs, I_j[s], and output states, x_j[s], will be set for each processing element.

For the output layer, the scaled local error from equation (2.7) and the delta weight of equation (2.10) will be calculated for each of its processing elements.

For the hidden layers, the scaled local error will be calculated using equation (2.11) and the delta weight using also equation (2.10).

An update of all the weights will be performed by adding these delta weights to their corresponding previous weights. [Ref. 4:pp. NC116]

III. ADAPTIVE CONTROL SYSTEMS

An adaptive controller is one in which the control system has the ability to adjust itself in order to compensate for some changes in the system's parameters and environment. Adaptive control is divided into two functions: a control function and a model estimation function.

The design of an adaptive control system can be conceptually simple when combining a particular parameter estimation technique with any control law.

In this chapter, two traditional adaptive control methods will be briefly discussed, a one step ahead control algorithm will be used to design the control model, and a linear least square estimation will be implemented with a recursive least-squares algorithm to produce a predictor-corrector equation used to design the estimator model.

A. TWO TRADITIONAL ADAPTIVE CONTROL METHODS

There are two traditional adaptive control methods that are of interest to neural network control theory: the self-tuning regulator of Astrom (STR) and the Lyapunov model reference adaptive control (MRAC). The self-tuning regulator block diagram is shown in Fig. 7. It is a classical feedback system with on-line adjustable coefficients. [Ref. 5]

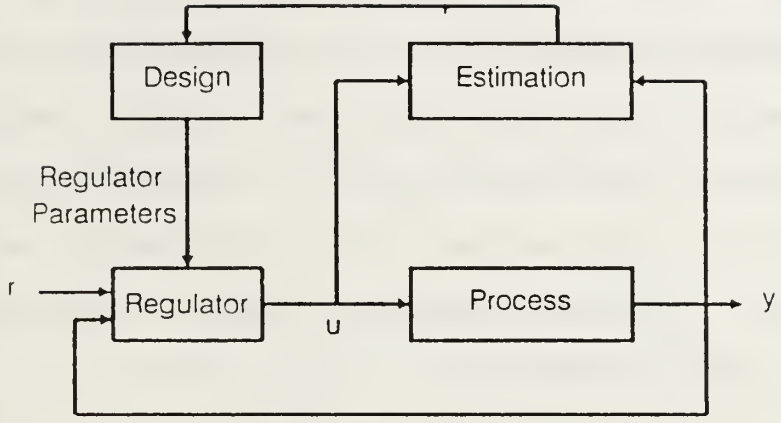


Figure 7 Self-Tuning Regulator Control Block Diagram

For the approximation of the model that describes the system being controlled, a least squares error parameter identification technique is used. The regulator parameters are adjusted during each control cycle according to the best estimate of the system parameters.

With this method, the stability of the system is not always guaranteed. During the learning phase the input signal can become infinitely large, thus the model is not realizable. If the least-squares error parameter estimates do not match entirely the description of the system, the closed-loop system performance will not satisfy the design specifications.

The second adaptive control method is the Lyaponuv model reference adaptive control (MRAC), shown in Fig. 8. The system is forced to follow a reference model. [Ref. 5]

The regulator consists of two loops, an inner and an outer loop. The inner loop is a feedback loop composed of the regulator and the plant.

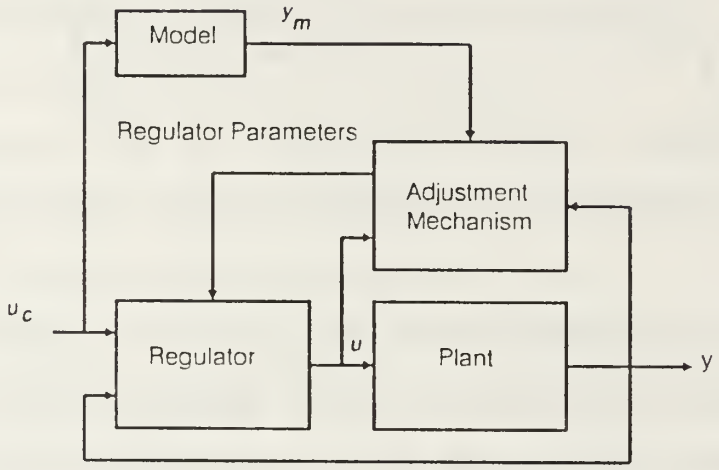


Figure 8 Model Reference Adaptive Control Block Diagram

The outer loop is also a regulator loop. It adjusts the parameters of the regulator by minimizing the error between the plant output y and the model output y_m. Therefore, the aim is to determine the adjustment mechanism so that the system being controlled tracks perfectly the reference model with zero error.

The latter type of adaptive controller will be emphasized in this thesis.

B. ONE STEP AHEAD PREDICTION CONTROL

The first function of adaptive control is control. The one-step-ahead controller is a very simple form of a control law. The basic idea is that the control input at each point in time is determined so as to bring the output, $y(t+d)$ (where d represents a time delay), to a desired output value, $y^*(t+d)$, in one step. This controller works not only for linear systems but also for a large class of nonlinear systems [Ref. 6:pp. 118-122].

The input-output properties of the system can be described by three equivalent model formats: a left difference operator representation, an observable state-space model, or a DARMA (discrete time deterministic autoregressive moving average) model [Ref. 6:p. 120]. The simplest to use for the development of adaptive control algorithms is the (DARMA) model. That model can be expressed as,

$$A(q)y(t) = B(q)u(t) \quad \text{where} \tag{3.1}$$
$$A(q^{-1}) = I+A_1(q)+\ldots+A_n(q)$$
$$B(q^{-1}) = B_o+\ldots+B_m(q)$$

where $A(q)$ and $B(q)$ are matrix polynomials expressed in terms of the backward shift operator, q^{-1}, the system output, $y(t)$, and the system input, $u(t)$. The terms in the past values of y are the *autoregressive* components and the terms in the past values of u are the *moving-average* components. A DARMA model

can be compared to a controllable and observable state-space model with an arbitrary initial state, or simply a transfer function. [Ref. 6:p. 32]

Rearranging equation (3.1) by expanding a single input single output (SISO) DARMA model in the shift operator gives,

$$y(t) = b_1u(t-1)+b_2u(t-2)+\ldots-a_1y(t-1)-a_2y(t-2)\ldots \quad (3.2)$$

which can be used to predict the output at the next time step,

$$y^*(t+1) = b_1u(t)+b_2u(t-1)+\ldots-a_1y(t)-a_2y(t-1)\ldots \quad (3.3)$$

where $y^*(t+1)$ is the predicted value of $y(t+1)$. The control input, u(t), of equation (3.3), which brings the system to a desired value $y^*(t+1)$ in one step, can be solved as follows:

$$u(t) = 1/b_1\ [y^*(t+1)+a_1y(t)+a_2y(t-1)+\ldots -b_2u(t-1)\ldots] \quad (3.4)$$

where the term $y^*(t+1)$ could be some reference input to the system [Ref. 1:p. 19].

The one step ahead prediction control law equation (3.4) minimizes the quadratic cost function comprising the squared prediction error:

$$J(t) = 1/2\ [y(t+1)-y^*(t+1)]^2 \quad (3.5)$$

Excessive effort may be required to bring $y(t+1)$ to $y^*(t+1)$ in one step. Therefore, some generalized cost functions of the same form like the *weighted one-step-ahead controller* [Ref. 6:p. 122], described below, could achieve a compromise between the level of effort expended and the prediction error.

If the past values of $y(t)$ and $u(t)$ are state variables, the one step ahead controller becomes,

$$u(t) = K(t)x(t)+r(t) \tag{3.6}$$

which is a state variable feedback with $r(t)$ as the reference input controller. From equation (3.4), the vector of past outputs and inputs provides the state variables in equation (3.6) which in turn provides a controller for an adaptive algorithm [Ref. 6:pp. 120-170]. This one step ahead controller could easily be modified to represent a weighted sum of state variables and a reference input, i.e., a weighted one-step-ahead controller,

$$u(t) = \Sigma_j W_{1j}*N_j(t) \tag{3.7}$$

where $N_j(t)=[\ r(t)\ u(t-1)\ u(t-2)...-y(t)\ -y(t-1)\ -y(t-2)...]$

As mentioned in the chapter of neural network theory, the input to a processing element is defined as the weighted sum of all the element activations coming to its input, as shown

in Fig. 2. Therefore, a direct comparison could be made between the two previous statements, in that this form of controller may well be represented by neural network processing elements.

C. LINEAR LEAST SQUARE ESTIMATION

The second function of adaptive control is estimation. On-line estimation techniques provide estimates for the system parameters based on minimizing the quadratic cost functions as in the case of a one-step-ahead controller.

The input-output characteristics of many linear and nonlinear deterministic systems may be described by the following model [Ref. 6:p. 50]:

$$y(t) = N(t-1)\theta^T \tag{3.8}$$

where $y(t)$ denotes the system output

$N(t-1)$ denotes a regression vector containing past measurements of the input and output,

θ denotes a parameter vector.

A first order DARMA model can be represented by [Ref. 6:p. 50],

$$y(t) = \Sigma\, b_j u(t-j) - \Sigma\, a_k y(t-k) \tag{3.9}$$

where j and k are indices for past input and output measurements. Equation (3.9) can be expressed in the form of equation (3.8) as follows:

$$N(t-1) = [u(t)\ u(t-1)\ u(t-2)\ldots -y(t-1) -y(t-2)\ldots] \quad (3.10)$$

$$\theta = [b_1 b_2 b_3 \ldots a_1 a_2 a_3] \quad (3.11)$$

With equation (3.9) the prediction error becomes,

$$\epsilon(t,\theta) = y(t) - N(t-1)\theta^T \quad (3.12)$$

where ϵ is used in the quadratic cost function to determine some optimal value for θ [Ref. 7:pp. 176-179],

$$J(\theta) = 1/2\ \Sigma_t[\epsilon]^2 \quad (3.13)$$

where t covers 1 to n measurements. Equation (3.13) can be minimized analytically by differentiating with respect to θ and setting the result equal to zero, which on solving for θ gives the *linear least square estimate*,

$$\Theta = [1/n\ \Sigma_t N(t) N^T(t)]^{-1}\ 1/n\ \Sigma_t N(t) y(t) \quad (3.14)$$

where Θ is the estimated parameter vector, equation (3.11).

The recursive least-squares algorithm [Ref. 7:p. 307]

summarizes in five equations the way of determining the parameter vector, θ,

$$\begin{aligned} \Theta(t+1) &= \Theta(t)+L(t)[y(t)-\Theta^T(t)N(t-1)] \\ L(t) &= P(t-1)N(t-1)[1+N^T(t-1)P(t-1)N(t-1)]^{-1} \\ P(t) &= P(t-1)-[A(P,N,t)/B(P,N,t)] \\ A(P,N,t) &= P(t-1)N^T(t-1)N(t-1)P(t-1) \\ B(P,N,t) &= 1+[N^T(t-1)P(t-1)N(t-1)] \end{aligned} \tag{3.15}$$

The first equation of (3.15) is a predictor-corrector equation, while the others solve for the estimation gain, $L(t)$.

A special form of a predictor-corrector equation that is used in many least-squares parameter estimation applications is expressed as follows: [Ref. 6:p. 49]

$$\theta(t+1) = \theta(t)+M(t)N(t-1)e(t) \tag{3.16}$$

where $\theta(t)$ denotes the parameter estimate at time t

$M(t)$ denotes an algorithm gain (possibly a matrix)

$N(t-1)$ denotes the regression vector

$e(t)$ denotes the model prediction error

The gain term, $M(t)$, may vary from a scalar constant to a covariance matrix, as seen in equation (3.15) with $L(t)$.

From the back-propagation learning rule discussed in the previous chapter, equation (2.10) is very similar to the linear least squares parameter estimator, equation (3.16),

$$\Delta w_{ji}^{[s]} = lcoef * e_j^{[s]} * x_i^{[s-1]} \qquad (2.10)$$

The learning coefficient, *lcoef*, is equivalent to the algorithm gain, $M(t)$; the error signal, e_j, is equivalent to the model prediction error, $e(t)$; and the activation value, x_i, is equivalent to the regression vector, $N(t-1)$. Thus, a neural network using the back-propagation algorithm is in a sense a linear least squares estimator [Ref. 7:p. 22]. The algorithms and theorems applicable to the linear least squares estimation should, for the most part, be applicable to the back-propagation neural network.

In summary, adaptive control is a combination of a control method and a model estimation. For the control method, the Model Reference Adaptive Control (MRAC) was chosen over the Self-Tuning Regulator (STR). As to the controller itself, the weighted one step ahead prediction controller was defined as the weighted sum of the state variables and the reference input. These weights or feedback gains are extracted from the adjustment mechanism of MRAC in Fig. 8 and are determined by minimizing the error between the network predicted output and the model output.

For the estimation model, which is the linear least squares estimate, the predicted output was also defined as some weighted sum of the terms in the regression vector. This time, the weights are determined using the predictor-corrector equation to minimize the error between the measured and predicted output.

By combining the control and estimation models above, it appears that the back propagation learning rule could produce models for adaptive control problems.

IV. X-29 MODERN AIRCRAFT

The purpose of this chapter is to present a description of the modern aircraft chosen and to describe the H_2 and H_∞ controllers that were previously designed to solve the instability of that aircraft. The model used is the subsonic longitudinal dynamics of the X-29 fighter aircraft. The X-29 is a single seat forward swept wing (FSW) demonstrator aircraft built by the Grumman Corp. The FSW design offers a new generation of tactical aircraft that is smaller, lighter in weight, less costly, and highly efficient.

A. FIGHTER DESCRIPTION

The aerodynamic advantages of a forward swept wing have been known since the 1940's. These advantages are: improved maneuverability with spin-proof characteristics, better low-speed handling, and reduced stalling speed. Another important advantage is the low drag across the entire operational envelope, particularly around the sonic speed, which permits the use of a less powerful engine [Ref. 8]. The FSW design is expected to reduce the transonic drag about 20%, and to handle low speeds and high angles of attack much better than any aft-swept wing [Ref. 8:pp. 47]. However, no suitable structure could be found in the 1940's to take full advantage of these benefits. Only years later, did the use of advanced

composite materials offer a solution. The graphite composite material, strong and light in weight, was utilized on the FSW to eliminate the adverse static aeroelastic coupling between wing bending and torsion. The X-29, shown in Figure 9, features close-coupled canard surfaces in front of the forward-swept wings for primary pitch control. [Ref. 9]

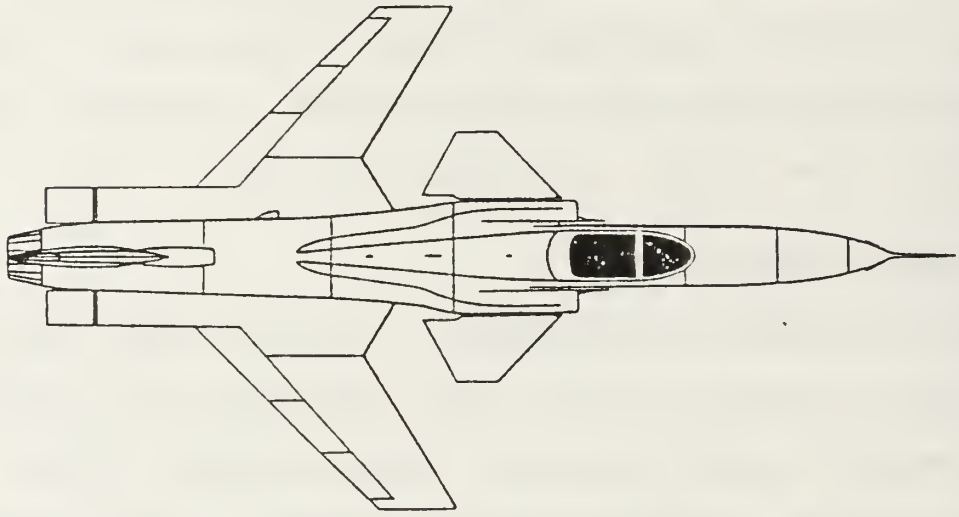


Figure 9 Grumman X-29 FSW Demonstrator Aircraft

The small strake flaps at the rear of the aircraft provide additional pitch at low speed.

To optimize the wing for various flight conditions, the wing's two segment trailing edges (flaperons) behave as a variable camber device for pitch control. This variable camber control will permit the airplane to be optimized for low

speed, maneuvering, cruising, and high-speed flight conditions.

The X-29 longitudinal dynamics model considered in this thesis is the analog reversion mode with the aircraft trimmed at 0.5 mach, 30,000 feet. The original 83rd order model was reduced to a 14 state model. The reduced model includes a short period approximation of the aircraft longitudinal dynamics, the vertical velocity, w, and pitch rate, q, and the fourth order actuator dynamics for the three longitudinal control surfaces, i.e., the canards, flaperons and strakes.

Figure 10 shows the open loop actuator/aircraft dynamics model of the X-29 [Ref. 2]. The two separated commands, r_1 and r_2, are the input to the three control surface actuators with r_1 controlling the canards and r_2 controlling the flaps and strakes. The outputs of the system are the two states, w and q. The w-state becomes the angle of attack α when divided by the initial forward velocity U_o. Therefore, the uncompensated model has two inputs, two outputs, and 14 states. The control inputs to the aircraft dynamics are the canards δ_c, the flaperons δ_f, the strakes δ_s and their respective first and second derivatives.

The uncompensated state variables are listed in Table I, and the open loop poles are listed in Table II. Notice the positive pole, 1.9550, on the real axis, meaning that the X-29 has an unstable short period mode.[Ref. 2]

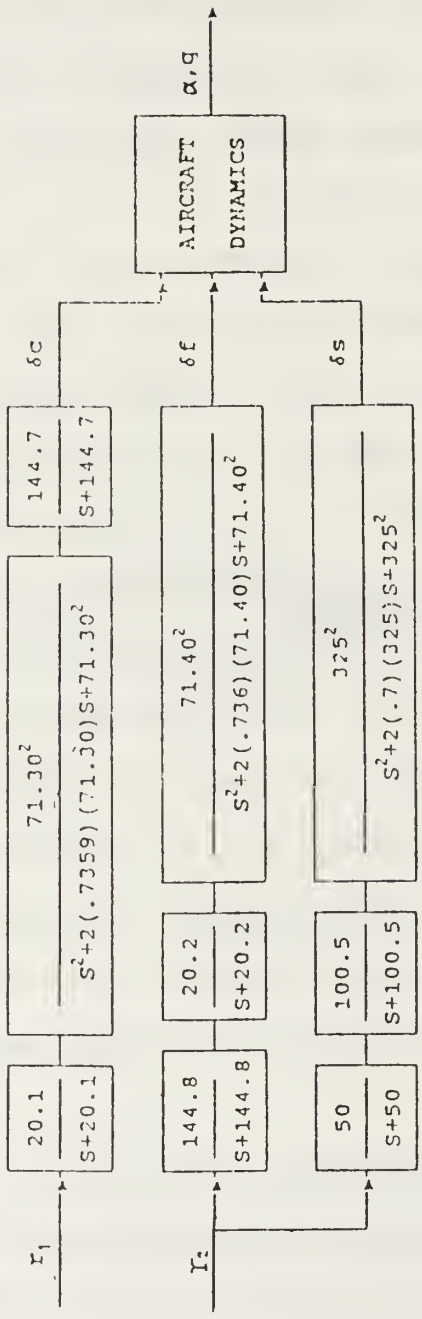


Figure 10 Uncompensated X-29 Open-Loop Plant

TABLE I : UNCOMPENSATED X-29 MODEL STATES

State	Description	Units
α	angle-of-attack	rad
q	pitch rate	rad/sec
δ_c	canard control input	rad
δ_f	flap control input	rad
δ_s	strake control input	rad
$\dot{\delta}_c$	canard control rate	rad/sec
$\dot{\delta}_f$	flap control rate	rad/sec
$\dot{\delta}_s$	strake control rate	rad/sec
$\ddot{\delta}_c$	canard control accel.	rad/sec^2
$\ddot{\delta}_f$	flap control accel.	rad/sec^2
$\ddot{\delta}_s$	strake control accel.	rad/sec^2
$\dddot{\delta}_c$	canard control jerk	1e+04 rad/sec^3
$\dddot{\delta}_f$	flap control jerk	1e+04 rad/sec^3
$\dddot{\delta}_s$	strake control jerk	1e+04 rad/sec^3

TABLE II : UNCOMPENSATED X-29 OPEN-LOOP POLES

-2.2746e+02 ± 2.3201e+02i
-1.4491e+02
-1.4455e+02
 1.9550e+00
-1.0031e+02
-2.7155e+00
-5.2506e+01 ± 4.8410e+01i
-5.2518e+01 ± 4.8255e+01i
-5.0067e+01
-2.0172e+01
-2.0115e+01

B. H_2 AND H_∞ CONTROLLERS

The uncompensated X-29 model possesses poor disturbance attenuation, high sensitivity to plant variations and modeling errors, and a small control bandwidth.

Three weighing functions were utilized in Ref. 2 to improve these performance characterizations by suppressing the sensitivity function singular values as much as possible, i.e., to make the loop gains as large as possible over a wider bandwidth. The resultant X-29 augmented plant is a 16th order system, in which the weighing functions added two states.

The two Riccati solution methods mentioned in Ref. 2 indicate that H_2 and H_∞ controllers must be the same size as the augmented plant. Therefore, the X-29 controller has to be of 16th order. As a result, the H_∞ compensated X-29 has a larger disturbance attenuation, lower sensitivity to variations and modelling errors, and a wider control bandwidth. [Ref. 2]

Two H_∞ and H_2 controllers have been designed to represent the optimal-performance and the limited- performance models.

The closed-loop architecture for the H_∞ compensated X-29 is shown in Fig. 11. The first block is the 16th order controller and the second block the 14th order plant matrix. [Ref. 2:p. 62]

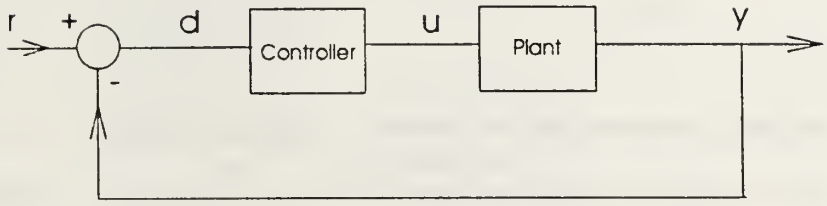


Figure 11 Closed-Loop Model of the H_∞ Compensated X-29

Unlike the open loop actuator/aircraft dynamics model of Fig. 10, the command vector r, composed of elements r_1 and r_2, represents the reference commands for the controlled outputs, α and q. The H_∞ controllers have been placed in series with the fighter plant to be fed backward with a negative gain of one. Therefore, the closed-loop system has two inputs, two outputs and thirty states.

The longitudinal equations of motion may be expressed in the following state variable form:

$$\dot{x}(t) = A\underline{x}(t) + B\underline{u}(t) \qquad (5.1)$$
$$y(t) = C\underline{x}(t) + D\underline{u}(t)$$

where $\underline{x}(t)$ contains the state variables of the H_∞ controller and of the X-29 plant,

$\underline{u}(t)$ contains the input variables $r_1(t)$ and $r_2(t)$, and

$\underline{y}(t)$ contains the output variables $\alpha(t)$ and $q(t)$.

1. Optimal-Performance Model (Compensated)

The Matlab program used to obtain the closed-loop state space representation of the H_∞ optimal-performance model is described in Ref. 2 pages 115 to 121.

The poles of the closed-loop model are listed in Table III. The unstable short period pole, 1.9550, of the open-loop system in Table II is mirrored into the left half plane in Table III.

Safonov [Ref. 11] indicates that this mirror imaging does not cause any limitation to the system's performance if this pole is not the dominant one.

The compensated X-29 model provides precision flight path control modes due to the multiple, independently controlled surface configuration. Figure 12 shows a graphic representation of these precision control modes,

where α denotes angle-of-attack,

θ denotes pitch attitude,

γ denotes flight path angle,

TABLE III : X-29 OPTIMAL PERFORMANCE CLOSED-LOOP POLES

-4.1327e+02
-2.2745e+02 ± 2.3201e+02i
-1.3068e+02 ± 4.6111e+01i
-5.3305e+01 ± 8.9700e+01i
-1.4491e+02
-1.4452e+02
-1.3014e+02
-1.3877e+01 ± 5.9243e+01i
-9.9794e+01
-5.2545e+01 ± 4.8359e+01i
-5.2503e+01 ± 4.8301e+01i
-7.4132e+01
-1.9550e+00
-2.7155e+00
-2.0379e+01 ± 2.1564e+01i
-2.0578e+01 ± 1.8907e+01i
-4.9199e+01 ± 6.5316e+00i
-4.2465e+01
-2.0184e+01
-2.0110e+01

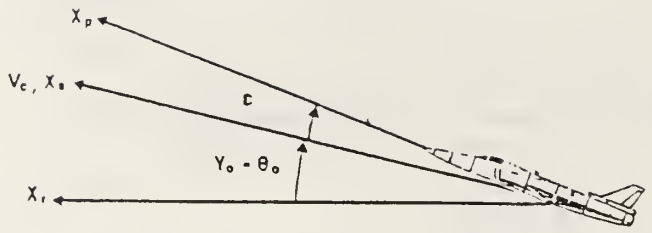
Xp and *s* denote the aircraft principal and stability axes.

The three precision longitudinal modes observed are [Ref.12]:

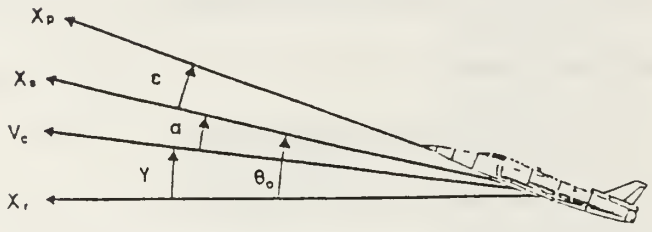
1. Vertical Translation: The aircraft's vertical velocity is controlled at a constant θ by varying α ,i.e., the aircraft's flight path angle, γ, or velocity vector, is controlled while *Xs* remains fixed.

2. Direct Lift Control: The aircraft's flight path angle, γ, is controlled at a constant α by varying θ, i.e., the aircraft's flight path angle, γ, or velocity vector, remains along the aircraft's axis *Xp* as *Xs* rotates.

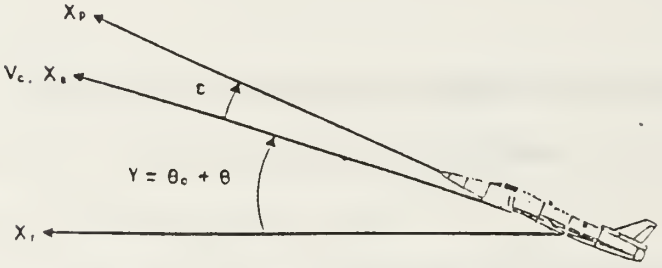
3. Pitch Pointing: The aircraft pitch attitude, θ, is controlled at a constant flight path angle, γ, i.e., the aircraft's flight path angle, γ, or velocity vector remains fixed while *Xs* rotates (θ=α).



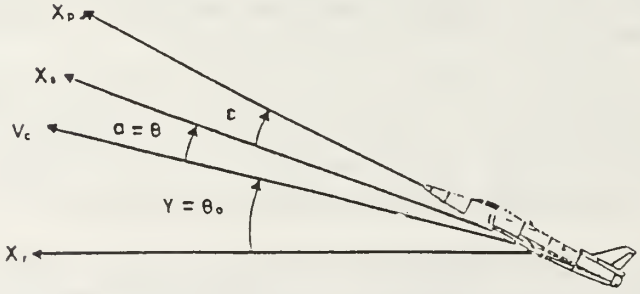
Reference Condition



Vertical Translation



Direct Lift Control



Pitch Pointing

Figure 12 Precision Control Modes

a. Time Domain

The longitudinal motion of the X-29 is described in time and frequency domains. For the time domain, a pulse input of one degree was applied for one second to each of the two reference commands. The $\alpha(t)$ and $q(t)$ time responses of the compensated X-29 for input 1, $r1$, and for input 2, r_2, are shown in Fig. 13 and 14.

The vertical translation mode is represented by Fig. 13 in which input 1 separates q and θ from α. It follows, that there are negligible changes in q (order of magnitude 10^{-1}) compared to α, which has a fast response of 0.180 sec.

The direct lift control mode is represented by Fig. 14, in which this time input 2 separates α from q. The X-29 responds to input 2 with a negligible α (order of magnitude 10^{-1}) and with a positive q rise time of 0.180 sec.

b. Frequency Domain

The continuous and discrete Bode frequency responses of $\alpha(t)$ and $q(t)$ for input 1, $r1$, and for input 2, $r2$, are shown in Fig. 15 through 18.

The discrete frequency responses are dominated by short period modes. For convenience, the high frequency dynamics above the nyquist frequency have been removed for the subsequent Bode plots.

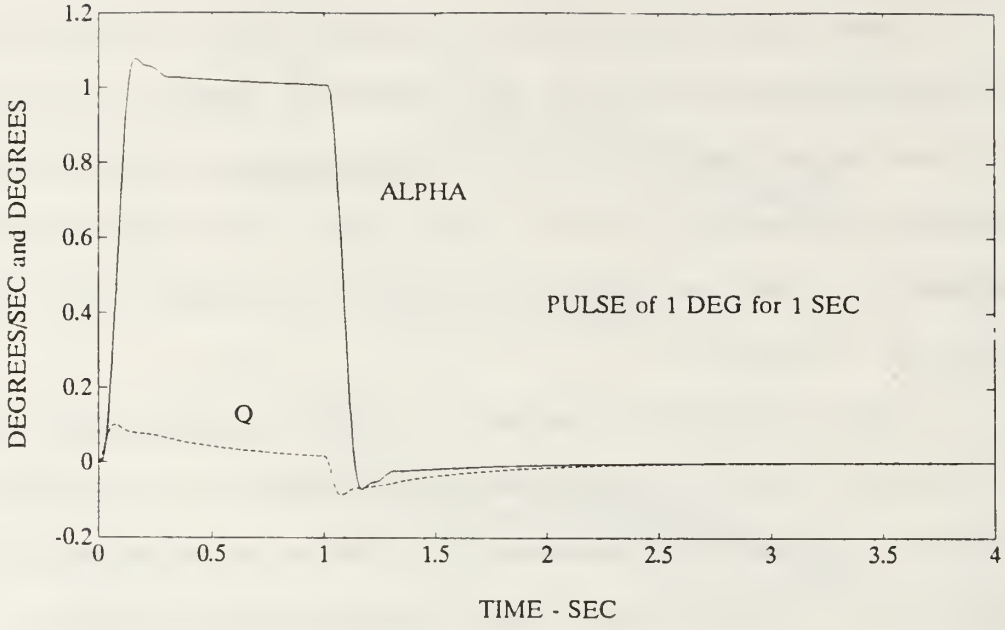


Figure 13 Q and α Time Responses to Input 1 (Optimal case)

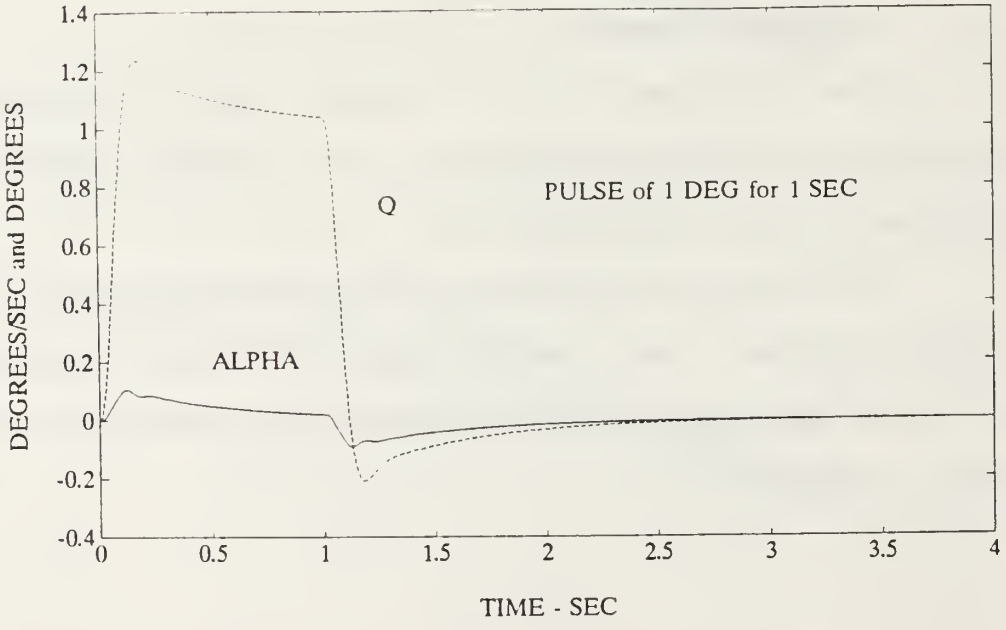


Figure 14 Q and α Time Responses to Input 2 (Optimal case)

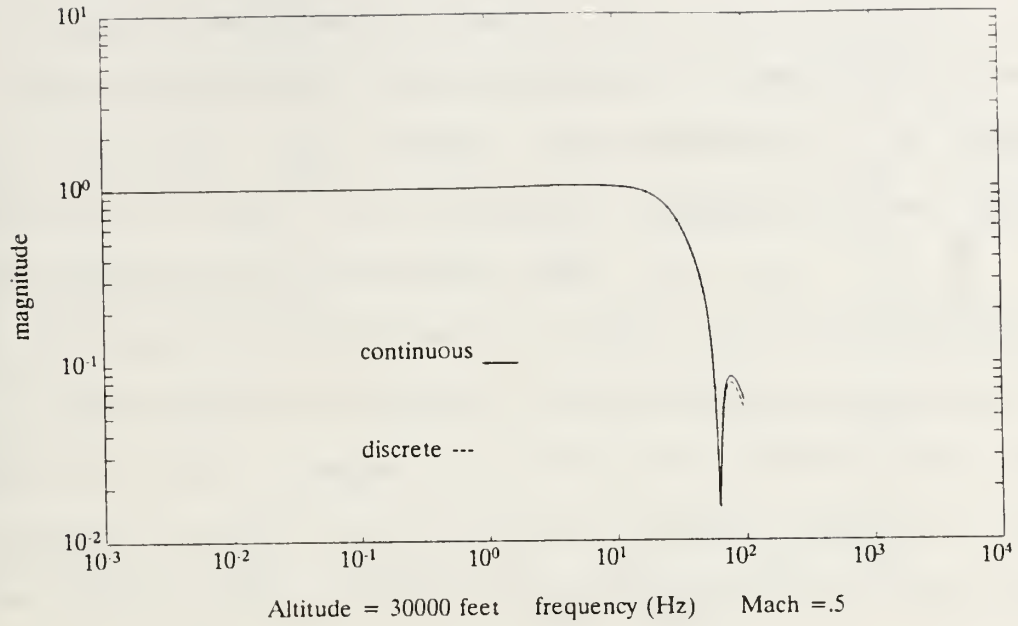


Figure 15 Continuous and Discrete α Frequency Responses to Input 1 (Optimal case)

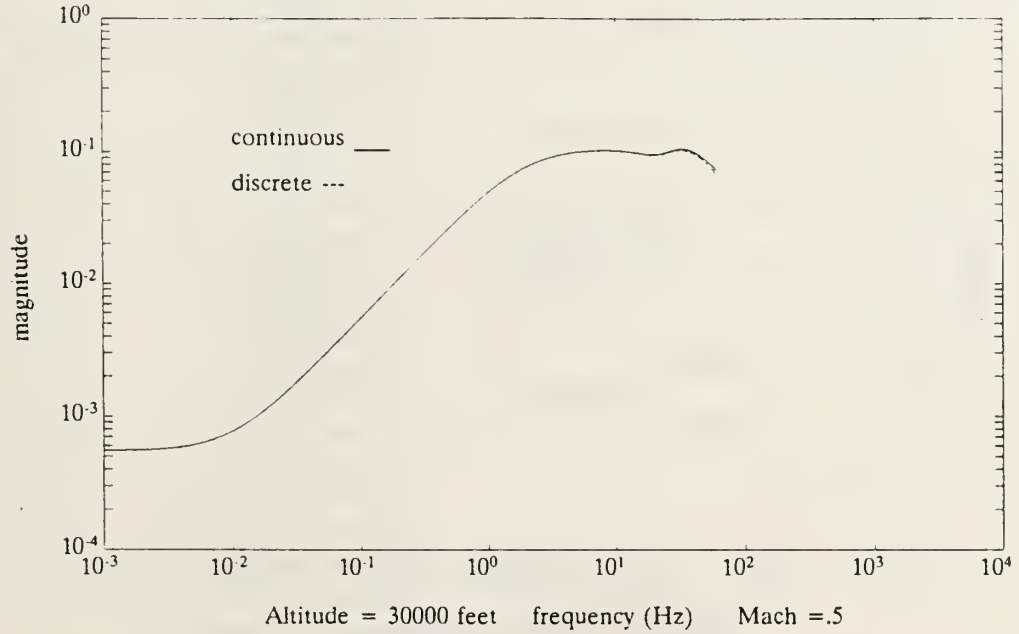


Figure 16 Continuous and Discrete q Frequency Responses to Input 1 (Optimal case)

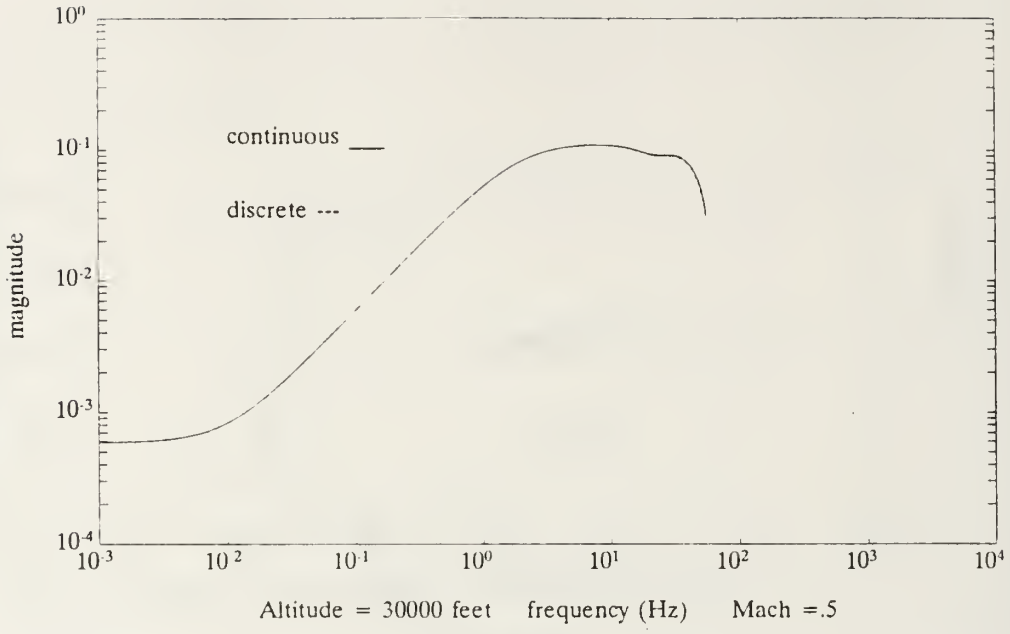


Figure 17 Continuous and Discrete α Frequency Responses to Input 2 (Optimal case)

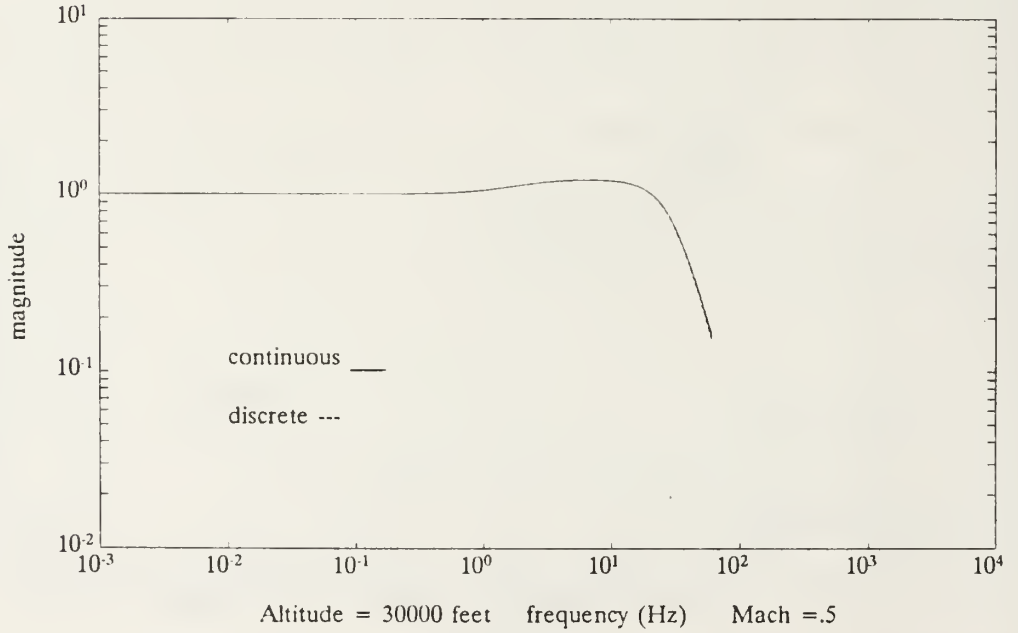


Figure 18 Continuous and Discrete q Frequency Responses to Input 2 (Optimal case)

2. Limited-Performance Model

The limited-performance X-29 can be characterized by a smaller control and closed loop bandwidth, a larger sensitivity to plant variations and modeling errors, and a smaller disturbance attenuation [Ref. 2:p. 87].

The Matlab program used to obtain the closed loop state space representation of the H_∞ limited-performance model is described in Ref. 2 pages 115 to 121. The closed loop poles are listed in Table IV. Notice that the same unstable short period pole, 1.9550, of the open loop system is mirrored into the left half plane.

TABLE IV : X-29 LIMITED PERFORMANCE CLOSED-LOOP POLES

-2.2747e+02 ± 2.3201e+02i
-1.3612e+02
-1.4415e+02
-1.4494e+02
-1.4476e+02
-1.0023e+02
-5.1688e+01 ± 7.7445e+01i
-7.2313e+01
-3.7606e+01 ± 5.2777e+01i
-5.1716e+01 ± 5.0560e+01i
-5.3298e+01 ± 4.7220e+01i
-5.2512e+01 ± 4.8306e+01i
-5.0505e+01
-3.6342e+01
-1.1889e+01 ± 1.2160e+01i
-1.9539e+00
-2.7204e+00
-3.7372e+00
-9.6752e+00
-2.0387e+01 ± 1.1180e+00i
-2.1180e+01

a. Time Domain

As in the optimal performance case, a step input of one degree for one second was applied to each of the two reference commands.

The $\alpha(t)$ and $q(t)$ time responses of the limited-performance X-29 for input 1, r_1, and for input 2, r_2, are in Fig. 19 and 20. The separation of α and q responses in both inputs are not as pronounced as in the compensated X-29 case. The step inputs with rise times of 0.5 sec and 0.8 sec to input 2 indicate that the limited-performance model was slower to react ,i.e., it is the result of a smaller closed loop bandwidth [Ref. 2:p. 93].

Therefore, the precision flight path modes in the case of the limited-performance X-29 are not as fully accomplished as the optimal-performance case.

b. Frequency Domain

The continuous and Bode frequency responses of $\alpha(t)$ and $q(t)$ for input 1 and 2 are shown in Fig. 21 through 24. As in the optimal performance model, the high frequency range was limited to 50 hertz.

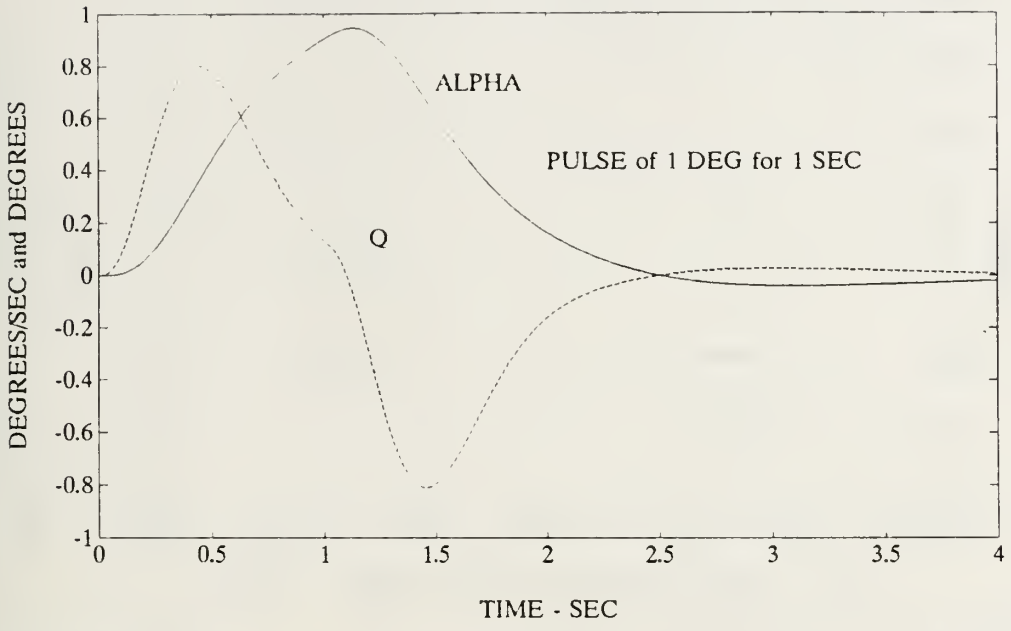


Figure 19 Q and α Time Responses to Input 1 (Limited case)

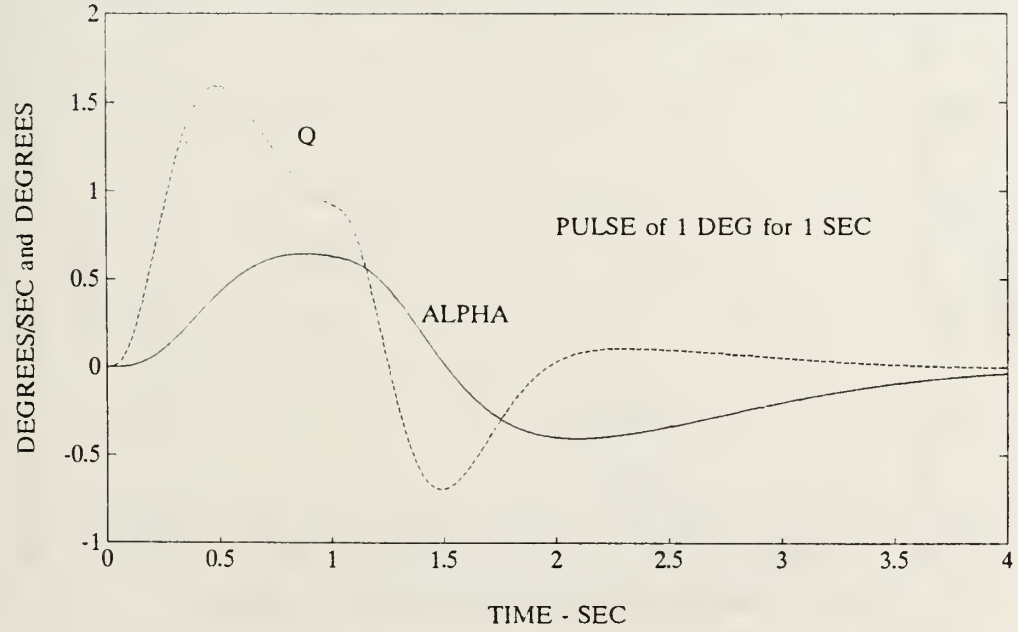


Figure 20 Q and α Time Responses to Input 2 (Limited case)

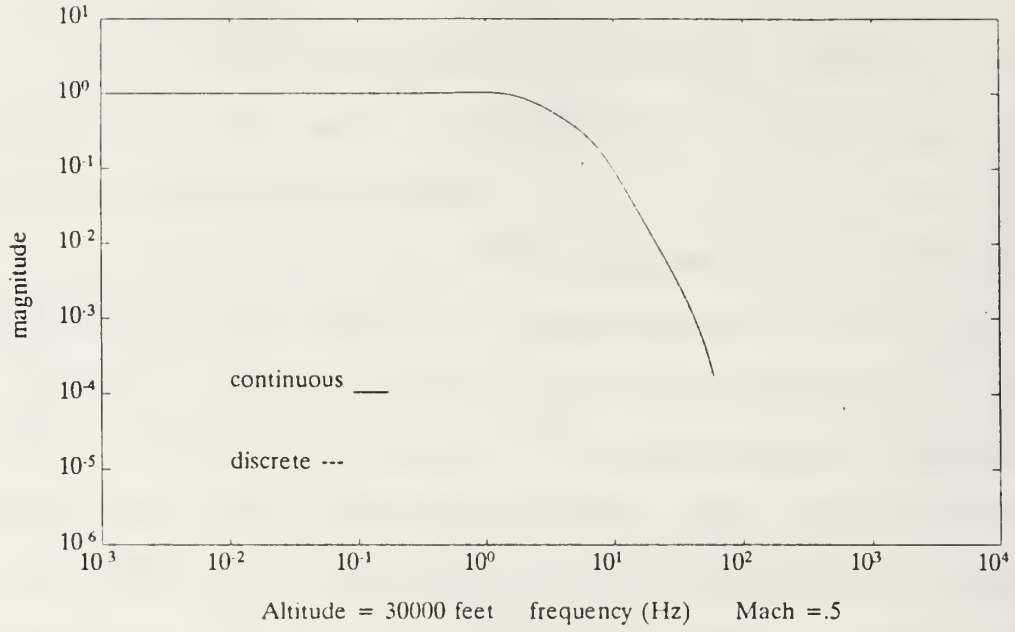


Figure 21 Continuous and Discrete α Frequency Responses to Input 1 (Limited case)

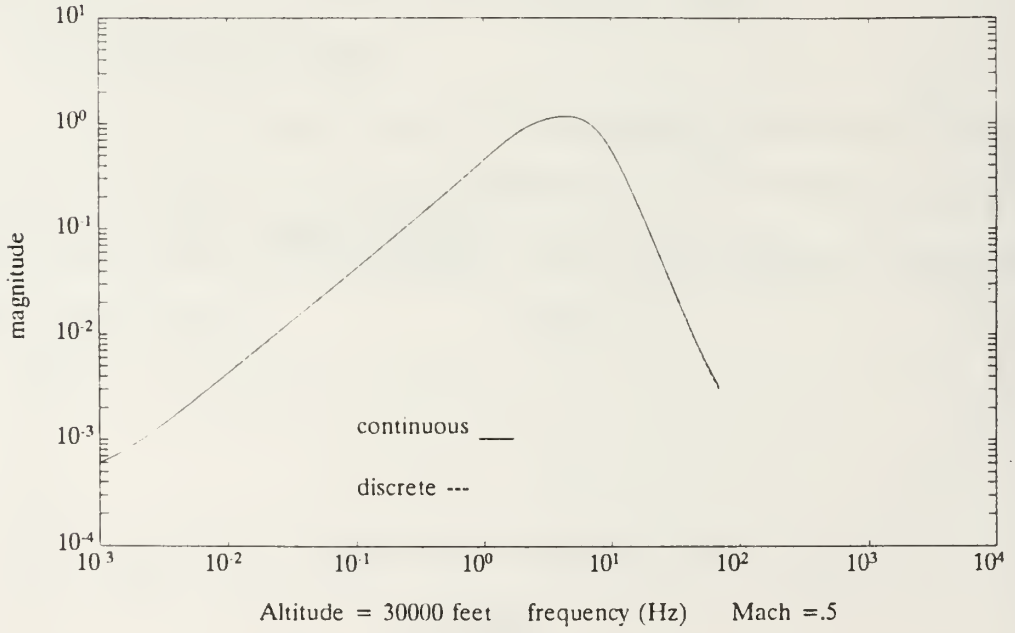


Figure 22 Continuous and Discrete q Frequency Responses to Input 1 (Limited case)

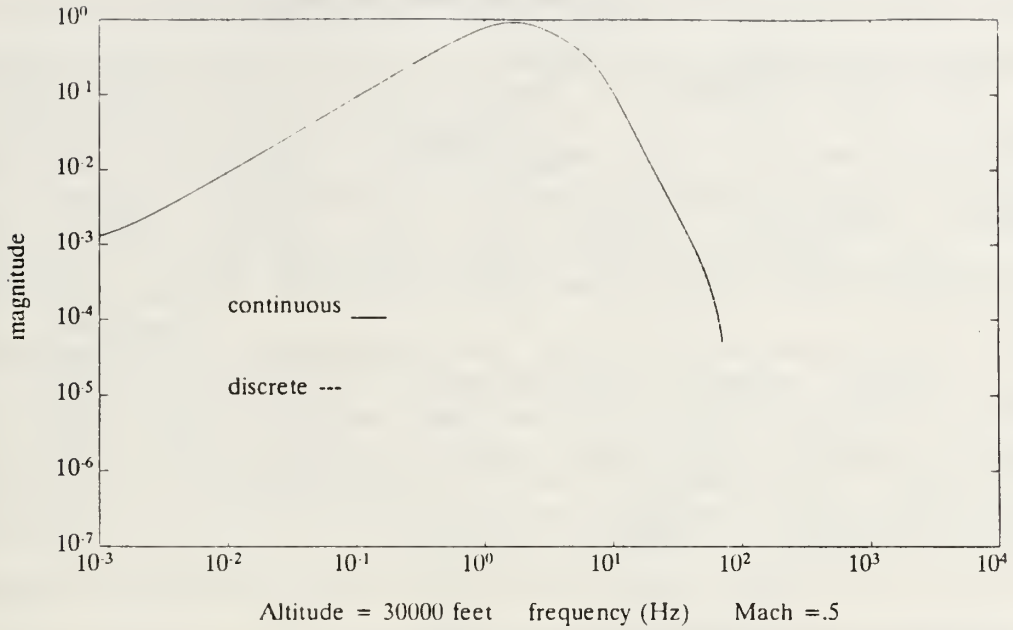


Figure 23 Continuous and Discrete α Frequency Responses to Input 2 (Limited case)

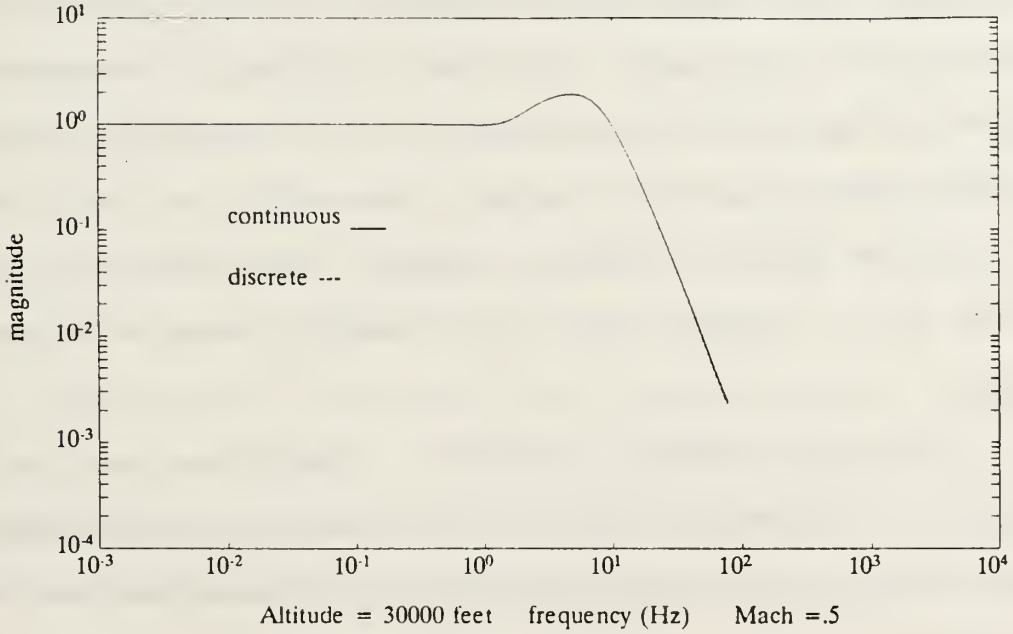


Figure 24 Continuous and Discrete q Frequency Responses to Input 2 (Limited case)

V. EXPERIMENTAL SET-UP

In this chapter, a description of the hardware and software used for the experimental set-up of the neural network adaptive control will be given. Model design considerations will follow which include design objectives, model structure selections, and choices of configurations. These configurations will be demonstrated with eight cases which are the framework of the experiments in the use of neural networks in adaptive control.

A. HARDWARE AND SOFTWARE REQUIRED

1. Hardware

All the research done in the area of data processing was conducted on the SPARC® Station 2. To emulate a parallel distributed processor which requires a very large memory capacity and high speed, the choice of this station becomes apparent. The SPARC® Station 2 uses a 32-bit architecture with a 40 mHz central processing unit, Sun 4/75 CPU, and a memory card with 48 mBytes of RAM. An internal hard disk drive of 207 mBytes and an external one of 996 mBytes were added to the system.

The workstation provides a multitasking windowed graphical environment, which greatly enhances the system's flexibility in addition to the powerful UNIX operation system, Sun OS 4.1.1.

A 16-inch high resolution color monitor, a 0.25-inch cartridge tape drive, and a 3.5-inch high density diskette drive are also part of the system.

Overall, the station's power and flexibility prove to be as important as its memory size and speed [Ref. 12].

2. Software

The Neuralworks Professional II/PLUS package by Neuralware, Inc was the neural networks software used. Neuralworks requires a minimum of 400 kBytes of base memory to operate, and offers over a dozen different types of networks, from the historical perception and the brain-state-in-a-box, to the back-propagation and the Boltzmann machines.

Neuralworks has a user's guide, some tutorials, a quick reference index, a menu listing, and many features of file inputs and outputs. It supports a general file format for inputting or outputting data to or from the network from standard spreadsheet file formats such as Lotus 1-2-3 and Excel, from the keyboard interface if formatted in ASCII files, or from user defined modules written in the C programming language. [Ref. 4:pp. UG215-UG250]

Neuralprobe instruments are available which provide the ability to perform internal network diagnostics by allowing specify information to be extracted from a selected probe, and by presenting that information in a graphical form for weights, error values, or activation levels.

As mentioned earlier, the capability of accessing the data internal to Neuralworks or presenting information to the network, is made possible through the use of a user defined module, USERIO. This method of access incorporates a user written procedure (SimoMonika.c) to the network, as shown in Appendix C.

The communication between Neuralworks and USERIO is through a series of data pointers which are described in the introductory pages of Appendix A.

Neuralworks uses user-defined control strategies to supervise the input-output sequencing, the learning, and how and when the information is passed through the layers of the network. Neuralworks provides also default control strategies for standard networks. The strategies are written in assembly like language and are automatically loaded into memory when the networks are loaded. The two control strategies used in this research are shown in Appendix D. Both strategies will be described in more detail in the following section.

The Matlab program with its signal processing and robust-control tool boxes was used intensively during the research. Written in C, Matlab provides a high-performance interactive software package for scientific and engineering computation [Ref. 13]. Matlab was employed mainly to perform time and Bode frequency response analysis by comparing the system and the actual neural network responses.

In summary, the SPARC Station 2, Neuralworks Professional II/PLUS, and the Pro-Matlab programs provided the necessary tools to successfully investigate the neural network adaptive control algorithm applied to the longitudinal dynamics of the X-29.

B. MODEL DESIGN CONSIDERATION

1. Design Objectives

In any implementation of a neural network adaptive controller, the design objectives of the system, the controller and the estimator, must be made clear.

The *system* must be controllable and observable in order for the controller and the estimator to be realizable. The Optimal and Limited performance cases fulfill both conditions.

The *controller* has to be able to track some sort of model reference or predicted output. Stability is also an important aspect of a controller. The poles of the transfer function have to be well within the unit circle for stability. Zeros that are outside the unit circle are non-minimum phase zeros. When the transfer function is inverted, these non-minimum phase zeros become unstable poles. The unstable inverse transfer function requires complex control devices for exact tracking [Ref. 1:pp. 32-49]. The neural network adaptive controller handles the non-exact tracking by determining the control gains in some least square sense.

The back-propagation learning rule performs this task with the connections' weights.

The *estimator* has to model the input-output relationships of the system. The selection of proper inputs is a complex issue involving the input spectrum, the sampling time, and the data record length.

a. Input Spectrum

All modes of the system must be excited, which is known as the concept of persistent excitation [Ref. 7:p. 72]. This concept can be best achieved by a proper selection of the input spectrum. The input spectrum must be selected in such a way that the output signal strength exceeds any expected noise. A high signal to noise ratio must be maintained to conserve most of the information content of the input signal.

b. Aliasing and Sampling time

Since sampling the data leads to information losses, it is important to select the proper sampling time. The information loss is best described in the frequency domain, where

$w_s = 2/T$ denotes the sampling frequency where T is the sampling interval, and

$w_N = w_s/2$ denotes the Nyquist frequency.

The part of the signal spectrum that corresponds to frequencies higher than w_N will be interpreted as a contribution from lower frequencies. This superposition is

known as "*aliasing*". Thus, the lost of information concerning frequencies higher than the Nyquist frequency is due to sampling. The best antialiasing filter is to sample fast enough to eliminate the high-frequency noise contributions.

Nevertheless, sampling too fast may cause loss of information in the low frequencies, while sampling too slow may cause loss of information in the high frequency modes.

Another problem of sampling too fast is the energy distribution problem in the higher frequencies which receive more excitation. This phenomenon will be demonstrated in Chapter VII with different plots of Bode frequency responses.

Figure 25 shows the resulting poles-zeros plot for the selected sampling time of 0.02 seconds. Notice that the poles and zeros are well distributed between z = 0.0 and z = 1.0, which is adequate for this investigation. Various trials were conducted with different sampling times but no improvements to the value of 0.02 seconds were found.

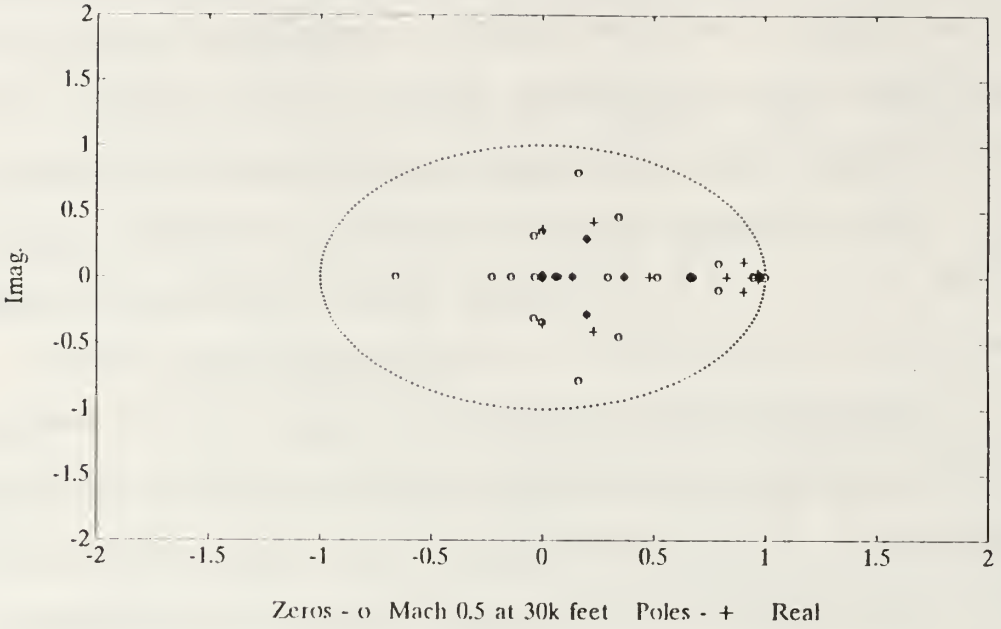


Figure 25 Poles and Zeros of the X-29 Closed-Loop Plant

c. Data Record Length

The simulation used in the USERIO program could generate data indefinitely. Modelling errors in the simulation like aliasing, the presence of non-minimum phase zeros and unstable poles propagate at a rate proportional to the power of the absolute value of the system zeros [Ref. 5:p. 43]. To prevent the modelling errors from growing unboundedly, the simulation has to be reset every so many cycles. Resetting the simulation adds noise to the frequency spectrum, as will be demonstrated in the next chapter.

For linear, stable systems, the data will be generated indefinitely since the errors will not grow unbounded. For linear, unstable systems, the data record

length will be determined by limiting the network outputs to specific values. Finally, for non-linear systems using an activation function like the hyperbolic tangent function, the network output values will be limited to $\pm$ 1.

2. Model Structure Selection

The model structure is based on the neural network adaptive controller in which the estimation and control algorithms are represented. Before selecting the model, the number of input and output elements, the status (linearity or non-linearity), and the order of the system have to be known. This will determine the size of the regression vector which defines the number of elements in each layer. Whenever simulating a nonlinear model, hidden layers are utilized with nonlinear transfer functions.

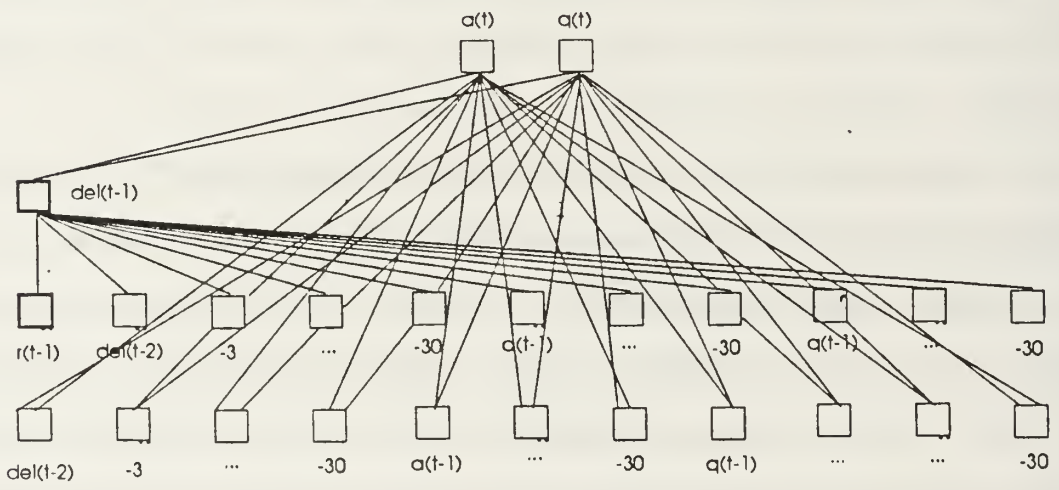
The closed-loop system representing either the optimal or the limited-performance, is a 30th order linear system with two inputs and two outputs. Hence, the selected model structure could be a MIMO (multiple inputs - multiple outputs) or a SIMO (single input - multiple outputs) if only one input is activated at one time when testing the network. The chosen network structure does not have any hidden layers since the system is linear. The network should possess a regression vector of 90 elements if represented by a SIMO model structure, or 120 elements if represented by a MIMO model structure (30 for each input and 30 for each output).

Two basic neural network model structures were developed for this research.

a. SIMO Neural Network Model Structure

Both, optimal and limited, performances could be represented with SIMO model structures. Figure 26 shows the SIMO neural network structure used for the 30 states closed loop X-29 plant. The first layer, the *feedback layer*,

SIMO Model Structure



consists of 89 elements. The first 29 elements are the past input values *del(t-2), del(t-3), ...del(t-30),* where the delay is indicated in parentheses. The remaining 60 elements represent the delayed past output measurements, 30 for $\alpha(t)$ and 30 for $q(t)$.

The *command layer*, the second layer, is a replicate of the first layer with the exception of the reference input, *r(t-1)*, which is needed for the control law.

The *control layer*, the third layer, consists of a single element, *del(t-1)*, the control input. The connections between the control input and the command layer elements are weighted with a fixed value of zero, since the command layer was intentionally included in the network only to represent the regression vector, i.e., no learning is taking place. For control law purposes, only the connection between the reference input, *r(t-1)*, and the control input, *del(t-1)*, is weighted with a fixed value of one. Therefore, *r(t-1)* equals *del(t-1)*.

The last layer is the *output layer*. The output layer is fully connected with variable weights to the control and the feedback layers.

Notice that this network has no hidden layers between the control and the output layers since all inputs are directly connected to the output layer. Nevertheless, the single element in the third layer becomes a hidden layer in itself, since it relays the outputs of layer two to layer four.

The estimation process begins when the activation value of each output element is compared directly to its model predicted output, and the current error is back-propagated through the control and feedback layers by adjusting their weights.

b. MIMO Neural Network Model Structure

Figure 27 shows a MIMO model structure used for the 30 state closed loop plant of the X-29. This time, two reference and control inputs were present, $r_1(t-1)$, $r_2(t-1)$, $del_1(t-1)$ *and* $del_2(t-1)$. As with the SIMO structure, $r_1(t-1)$ and $r_2(t-1)$ equal $del_1(t-1)$ and $del_2(t-1)$.

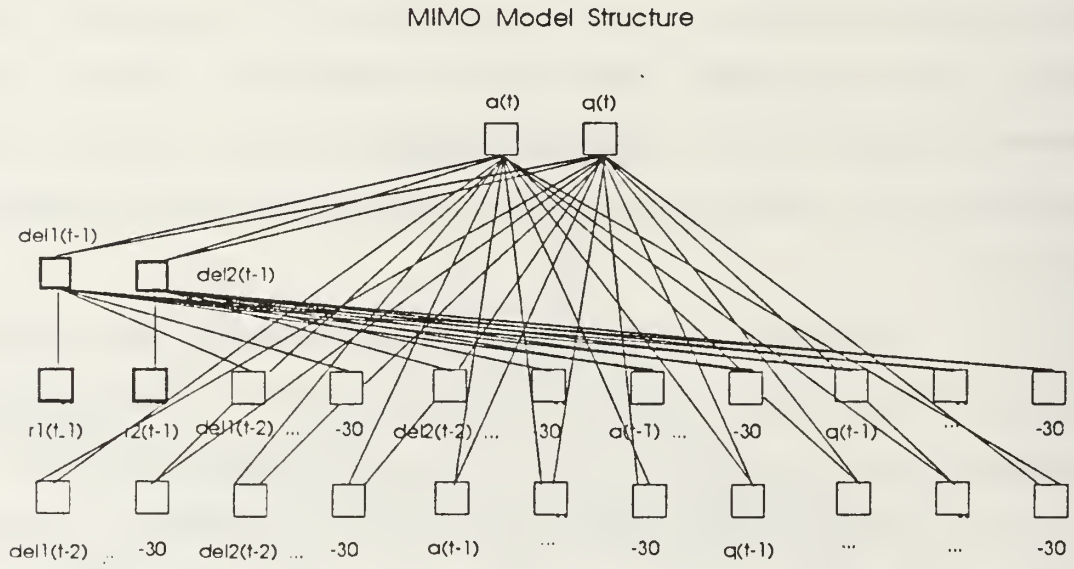


Figure 27 MIMO Neural Network Model Structure

As expected, the size of the regression vector has increased from 90 elements to 120, since the second input past measurements are added to the vector.

3. Choices of Configurations

In this thesis, three configurations are proposed for training neural networks to provide the appropriate inputs to

the X-29 plant in which desired responses are obtained. These three configurations are: the simulation of the closed-loop plant, the identification of the inverse plant, and the simulations of the existing controllers and the plant.

a. Simulation of the Closed-Loop Plant

In this first configuration, the neural network will emulate the closed-loop architecture of Fig. 28. The inputs, r_1 and r_2, to the 30 states transfer function that comprises the controller and the plant in series and a negative feedback loop of gain one, will be the inputs to the neural network. The outputs of the 30 states transfer function, α and q, which represent the true system outputs, will be the desired outputs of the neural network.

Closed-Loop Architecture

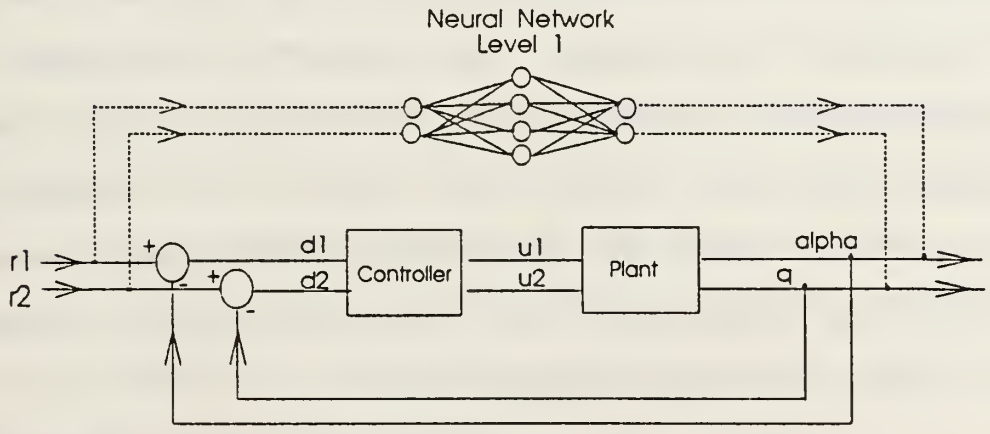


Figure 28 Closed-Loop Architecture

Each performance, the optimal and the limited, will be emulated using one MIMO model structure as shown in Fig. 27.

In order to facilitate the integration of the configurations to the USERIO program, specific case numbers are associated with each configuration. That way all three configurations can use the same USERIO program, SimoMonika.c. Each case number is stated in a header file. A header file defines all the variables utilized in the USERIO program: the case number, the sampling time, the input conditions, the numerator and the denominator coefficients of the transfer functions. All the header files, or transfer.h files, used for this research are shown in Appendix C. The numerator and denominator coefficients were obtained using the MATLAB file of Appendix D.

As to the control strategy, the first prototype, *contstr1.nnc*, described in Appendix B is employed when only one structure or network is needed to represent the configuration. The second control strategy, *contstr2.nnc*, is employed when two networks are needed to represent the configuration. Therefore, the single MIMO network needed to simulate the closed-loop plant should be trained using the first control strategy, *contstr1.nnc*.

Both control strategies use the back-propagation algorithm with the generalized delta learning rule which reduces the error between the actual and desired outputs of a

processing element by modifying the incoming connection weights.

The table in Appendix E shows the case number, the model structure, the control strategies, and the header files associated with each configuration. Notice that this first configuration contains two cases. Case #1 emulates the optimal performance model, and case #2 emulates the limited performance model. Also notice that the model structure has been divided into two levels. Level 1 and 2 represent the first and the second network to be trained. Both levels are usually connected in series and could be trained either one at a time or both simultaneously depending on the configuration chosen. In this configuration, only level 1 applies since only one network is necessary to represent each of the two cases.

When training neural networks, the stability of the system being emulated is an important factor to be considered. Since the transfer functions of the optimal and the limited-performance models are stable and linear, there should be no requirements for resetting the networks. The error should not grow unbounded during the learning process.

b. The Identification of the Inverse Plant

Figure 29a demonstrates the second configuration where an adaptive architecture is presented to identify the inverse of a plant [Ref. 14].

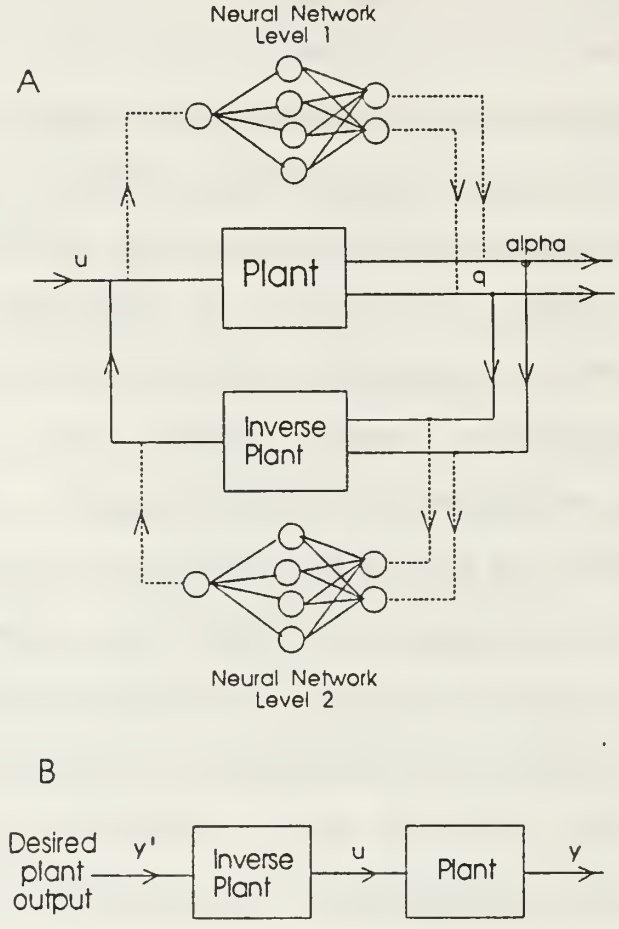


Figure 29 Inverse Plant Architecture

The single input, u, to the plant will be the input to the network emulating the plant, and the outputs to the plant, α and q, which represent the true plant outputs, will be the desired network plant outputs. Then, the network plant outputs become inputs to the network emulating the inverse plant, and the input to the plant, u, becomes the desired network inverse plant output. Once the plant inverse

has been found, it can be used for control purposes as shown in Fig. 29b. The desired plant output, $y' \approx y$, is fed into the inverse plant, and the resulting output is used as input to the plant. As a result, the plant input produces the desired plant output.

This control method can be applied to the control of linear and non-linear systems [Ref. 14:p. 34]. The configuration differs from the closed-loop architecture of Fig. 28 in that the input to the inverse plant is the desired plant output instead of the actual output, and that no feedback to the controller is required.

In this second configuration, three cases will be investigated, as shown in Appendix E. The first case examines the large order transfer function of the 30 states closed-loop X-29 longitudinal plant whose inverse is stable. The second case will examine a simpler aircraft, the A-4D, which has a much smaller order transfer function, but whose inverse is unstable. The inverse longitudinal plant of the A-4D aircraft is unstable due to the fact that the plant has a non-minimum phase zero at -3.65 whose inverse becomes an unstable pole, as shown in Fig. 30.

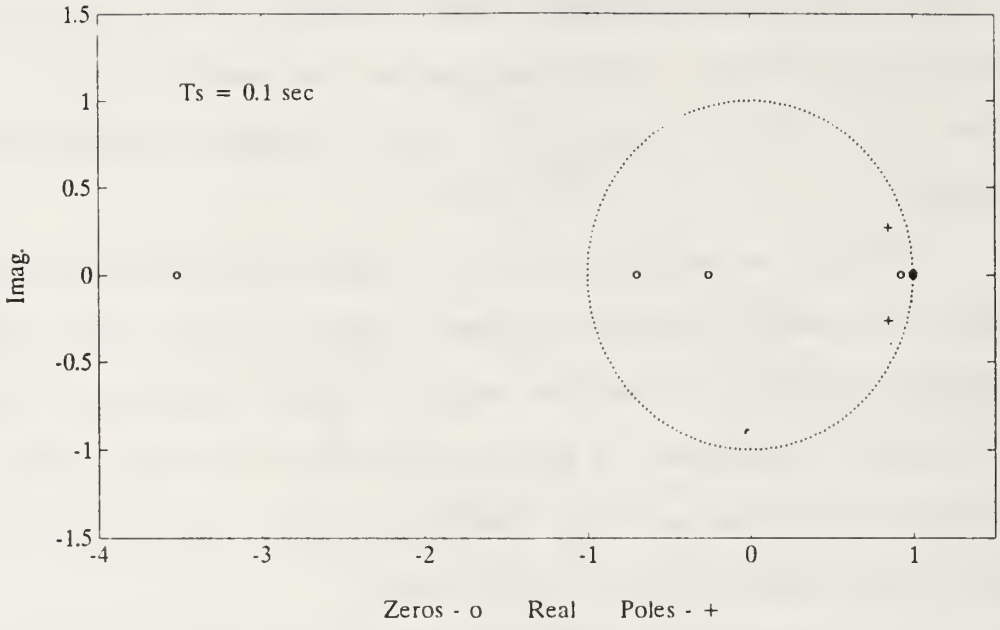


Figure 30 Poles and Zeros of the A-4D Plant

Finally, the third case examines the X-29 plant which is a larger order system than the A-4D and which has one unstable pole at +1.05 and three non-minimum phase zeros located at -4.5, +2.2 and +9.5, as shown in Fig. 31. As a consequence of the unstable pole, the X-29 plant is unstable, and as a consequence of the three non-minimum phase zeros, the inverse plant is also unstable.

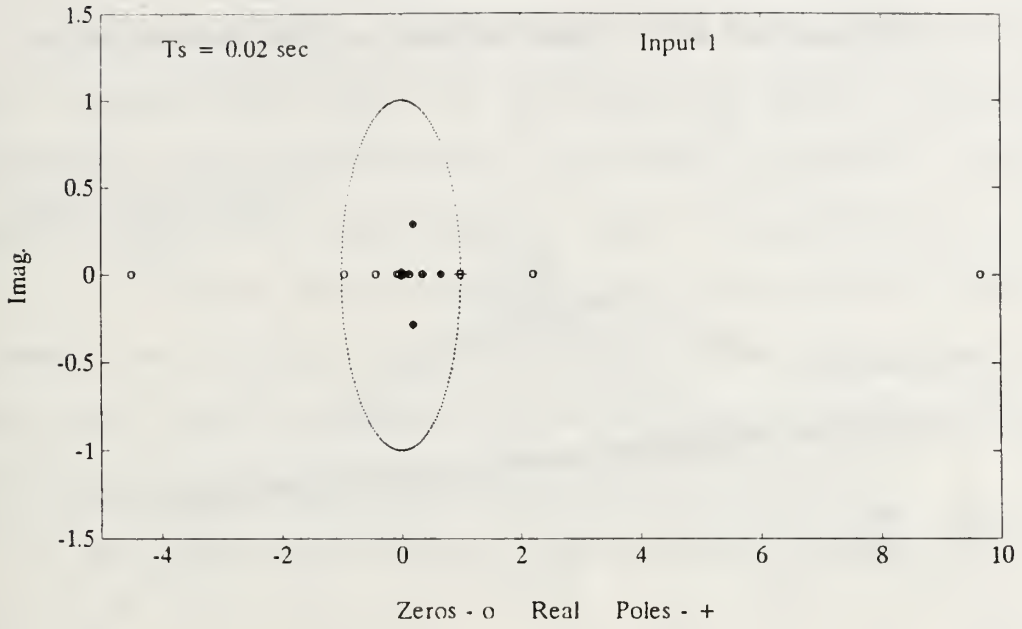


Figure 31 Poles and Zeros of the X-29 Plant

In all three cases, the model structure is composed of two levels: level 1 emulates the closed-loop plant for case #3 or the plant for cases #4 & 5, and level 2 emulates their inverse. The two levels are connected in series and can be trained simultaneously.

Figure 32 shows the inverse plant neural network structures developed for this investigation. As anticipated from Fig. 29, this configuration requires one SIMO neural network model structure to emulate the plant and one MISO neural network model structure to emulate the inverse plant.

Configuration # 2 : Identification of the Inverse Plant

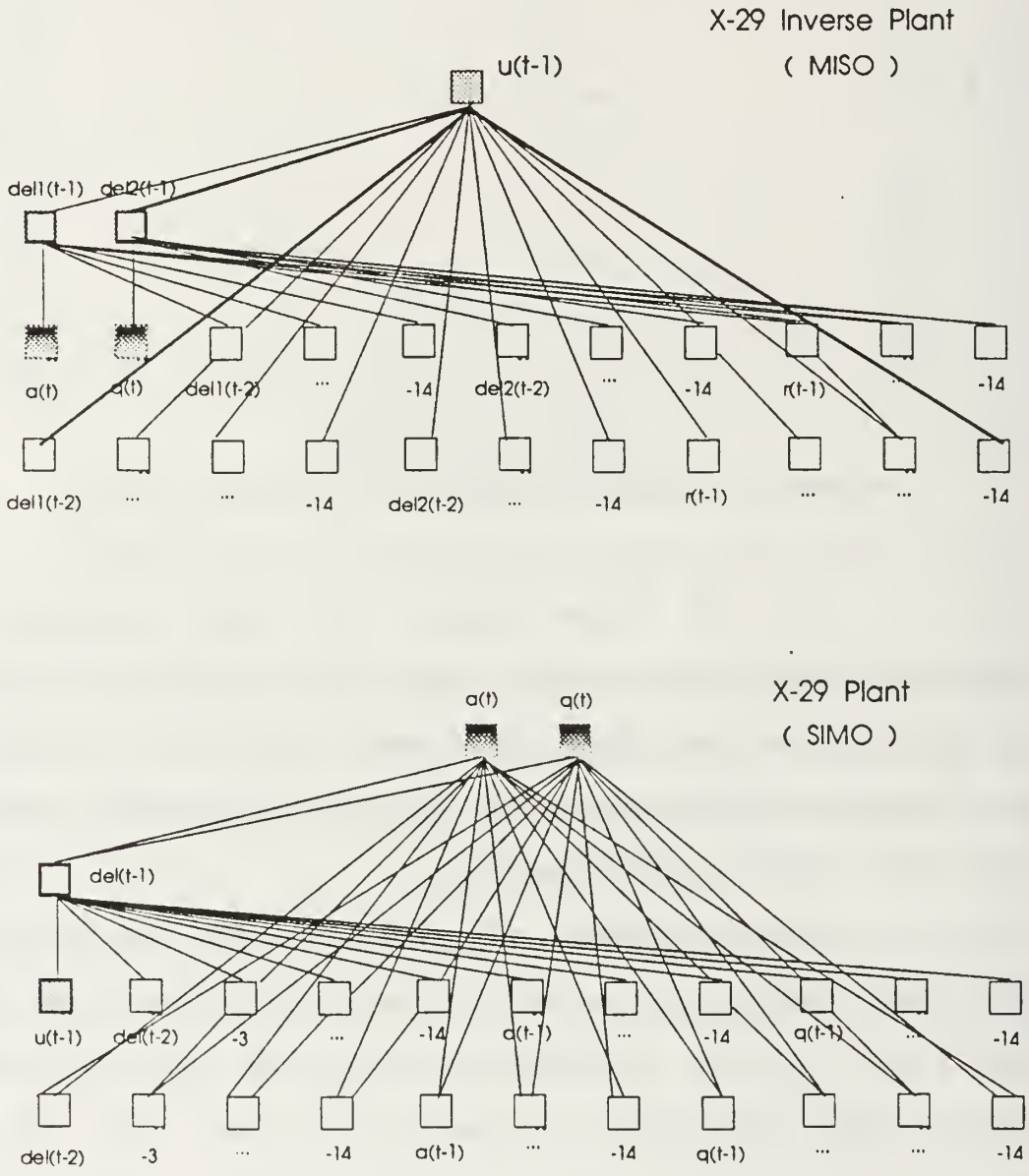


Figure 32 Configuration #2 : Identification of the Inverse Plant

The input of the plant, $u(t-1)$, becomes the output of the inverse plant, and the output of the plant, $\alpha(t)$ and $q(t)$, becomes the input of the inverse plant. These two operations are performed in the USERIO program, and are demonstrated in the introductory pages of Appendix A.

Since both systems, the X-29 plant and its inverse, are unstable, the neural networks will have to be reset many times to prevent the error from growing unbounded during learning.

c. The Simulations of the Existing Controllers and the Plant

Figure 33 shows the third configuration architecture. The inputs to the controller, d_1 and d_2, which are the error between the reference inputs and the plant

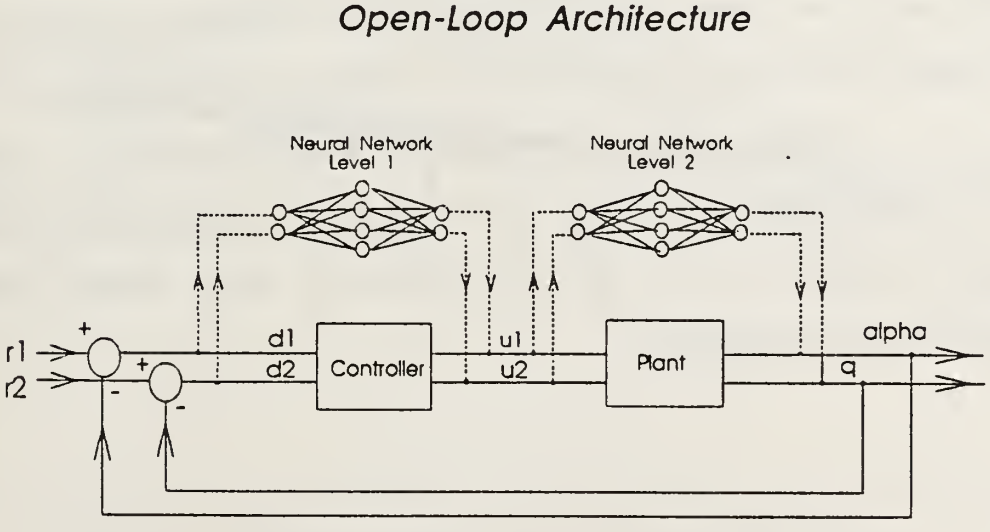


Figure 33 Open-Loop Architecture

outputs, will be the inputs to the network emulating the controller. The outputs of the controller, u_1 and u_2, will be the desired network controller outputs and the inputs to the network emulating the plant. The true plant outputs, α and q, become the desired network plant outputs.

In Fig. 28 the entire closed-loop architecture is emulated, whereas in this configuration each controller and plant is simulated separately as an open-loop architecture. Each performance, the optimal and the limited, has its own controller but the same plant.

As shown in Appendix E, the configuration has three cases. Case #6 & #7 emulate the optimal or the limited performance controller plus the plant, and case #8 makes the closure of the open-loop system of each case. The closure implies the connection in series of the controller and the plant, and the insertion of a negative feedback loop of gain one from the plant outputs to the controller inputs.

Both controllers are emulated at level 1 and the plant is emulated at level 2, using MIMO model structures as shown in Fig. 34. In all three cases both levels can be trained simultaneously.

Configuration # 3 : Simulation of the Existing Controllers and the Plant

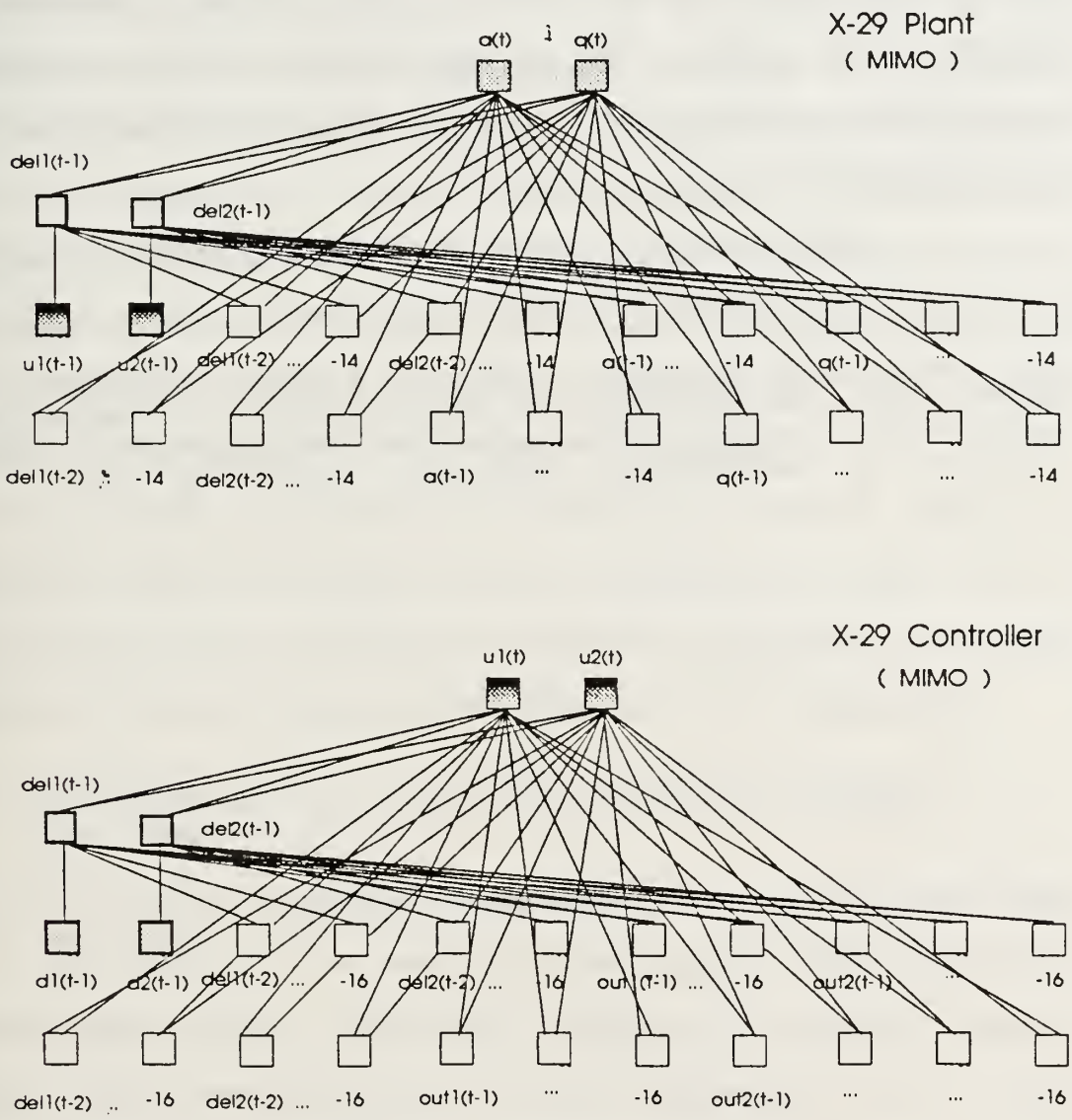


Figure 34 Configuration #3 : Simulation of the Existing Controllers and the Plant

Referring to Fig. 34, case #8 is represented by having the controller outputs, *u1(t)* and *u2(t)*, becoming the plant inputs, and the error between the plant outputs, $\alpha(t)$ and $q(t)$, and the reference inputs, *r1(t)* and *r2(t)*, becoming the controller inputs, *d1(t-1)* and *d2(t-1)*. These operations are performed in the USERIO program and are demonstrated in the introductory pages of Appendix E.

The utility of this third configuration, knowing that effective controllers exist, is that:

> The adaptive network may be able to form an effective control rule on the basis of representation of the system state that is easier to measure than the representation required by the existing controller. [Ref. 7:p. 30]

VI. RESULTS AND DISCUSSION

The experimental results of the eight configurations' cases described in Chapter V will be presented in this chapter. Time and frequency domain analyses of the neural network structures of each case will be performed to determine how close they are to the true system. All configurations except case #6 will use linear networks. Nonlinearity will be introduced in case #6, when two hidden layers are added with a hyperbolic tangent transfer function to the network emulating the optimal performance controller.

To further investigate the nonlinear network models, an analysis using the singular value decomposition (SVD) will be carried out on the controller network of case #6. The optimal number of elements per hidden layer will be determined.

A. CONFIGURATION #1: SIMULATION OF THE X-29 CLOSED-LOOP PLANT

1. Case #1 - Optimal Performance X-29 Closed-Loop Plant

The first case emulates the optimal performance X-29 closed-loop plant. The neural network is trained using a single MIMO model structure fully connected with a linear activation function. Fully connected means that all the elements in the feedback and control layer are connected to all the elements in the output layer. After 20,000 cycles or 300 seconds, the network has learned to respond correctly to

two random binary inputs of magnitude 1. The random binary swept square wave is an input signal which excites all the frequencies of interest.

The frequency responses of the linear neural network and the frequency responses of the optimal-performance X-29 model to two random binary inputs are displayed with discrete Bode plots in Fig. 35 through Fig. 38. The frequency responses of $\alpha(t)$ to input 1 and 2, which are shown in Fig. 35 and 37, develop near to exact model solutions. As expected, a small amount of unmodelled noise dynamics can be seen around the sampling frequency of 50 hertz. In Fig. 37, the network does not model exactly the low frequency region even after 20,000 epochs. The frequency responses of $q(t)$ to input 1 and 2, which are shown in Fig. 36 and 38, are very similar to the ones of $\alpha(t)$. The high frequency regions of both inputs and the low frequency region of input 2 are very well represented, whereas the low frequency region of input 1 shows a minor deviation from the true response, as shown in Fig. 36.

To show how close the network outputs are to the true X-29 outputs, the RMS prediction error plots for α and q are given in Fig. 39 and 40. Notice that the vertical scales are on the order 10^{-3}. As anticipated, the network has learned very well with RMS errors on the order of 0.00175 for α or 0.175 percent of the maximum output value of one, and on the order of 0.003 for q. Further training did not improve the present results.

The performance of the neural network in the time domain was determined by applying a step of one degree for one second to each input, and by comparing the network's outputs to the true system, as shown in Fig. 41 through Fig. 44. As with the frequency domain, the network has learned to model the true system in the time domain. By comparing the magnitude of the time responses of Fig. 41 with Fig. 42 , it can be seen that the X-29 model and the network respond to input 1 with a positive α while the q response is negligible (order of magnitude is 10^{-1}). In Fig. 41, the network responds with the same α rise time of 0.180 seconds, and with the same magnitude of about 1.08 degree as the true model. In Fig. 42, the network q response is as fast with a rise time of approximately .095 seconds. The small oscillations produced by the neural network q response in Fig. 42 are negligible since the magnitude of the signal is very low.

By comparing the magnitude of the time responses of Fig. 43 with Fig. 44, it can be seen that the responses of input 2 are the reversed responses of input 1, i.e., this time, the q response is positive while the α response is negligible with an order of magnitude 10^{-1}.

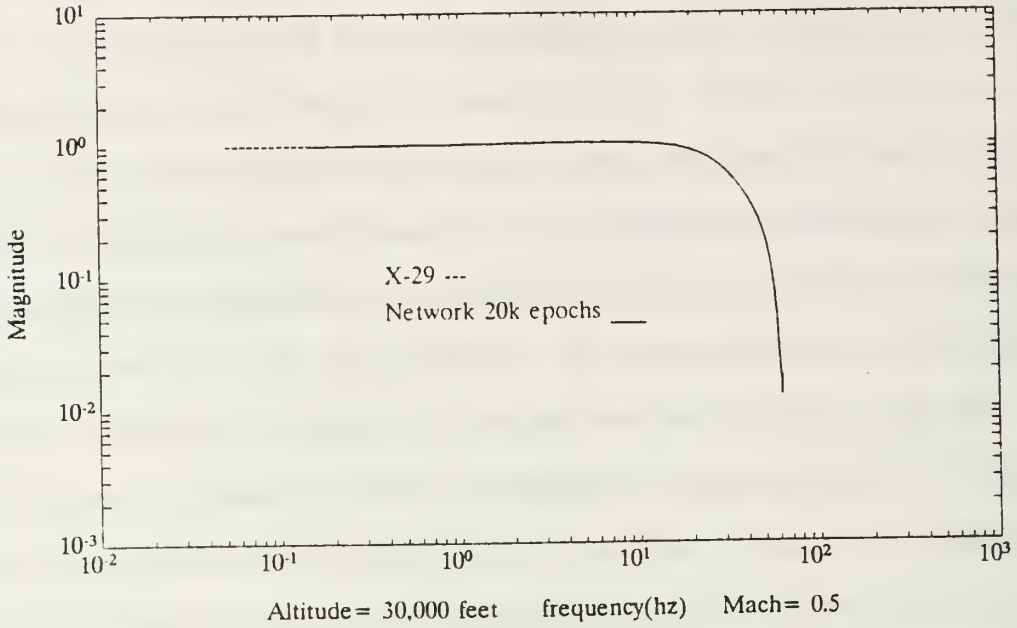


Figure 35 X-29 Model and Network α Frequency Responses to Input 1 (Optimal case)

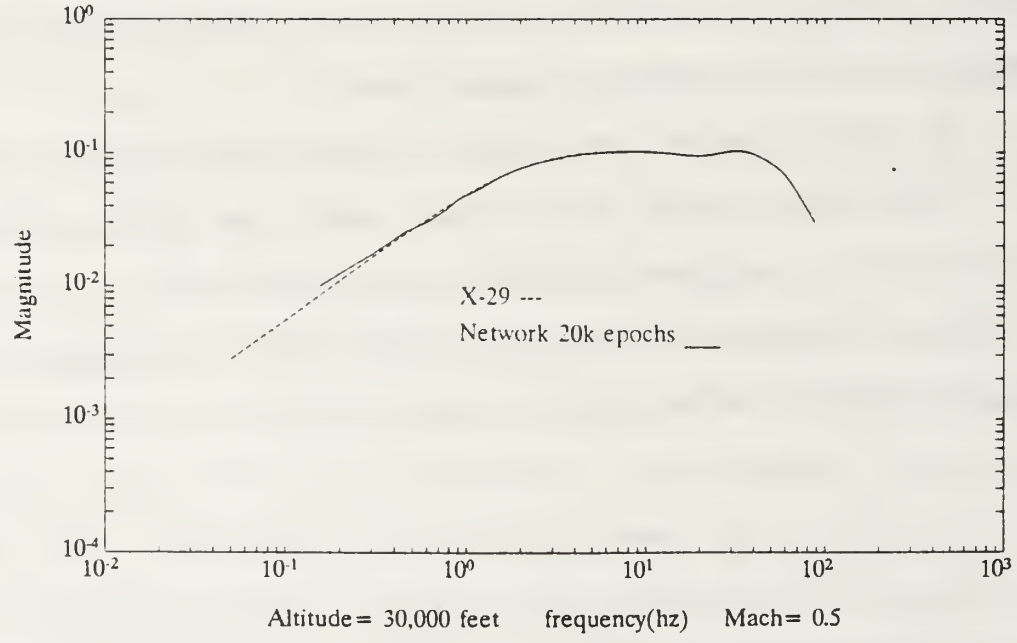


Figure 36 X-29 Model and Network q Frequency responses to Input 1 (Optimal case)

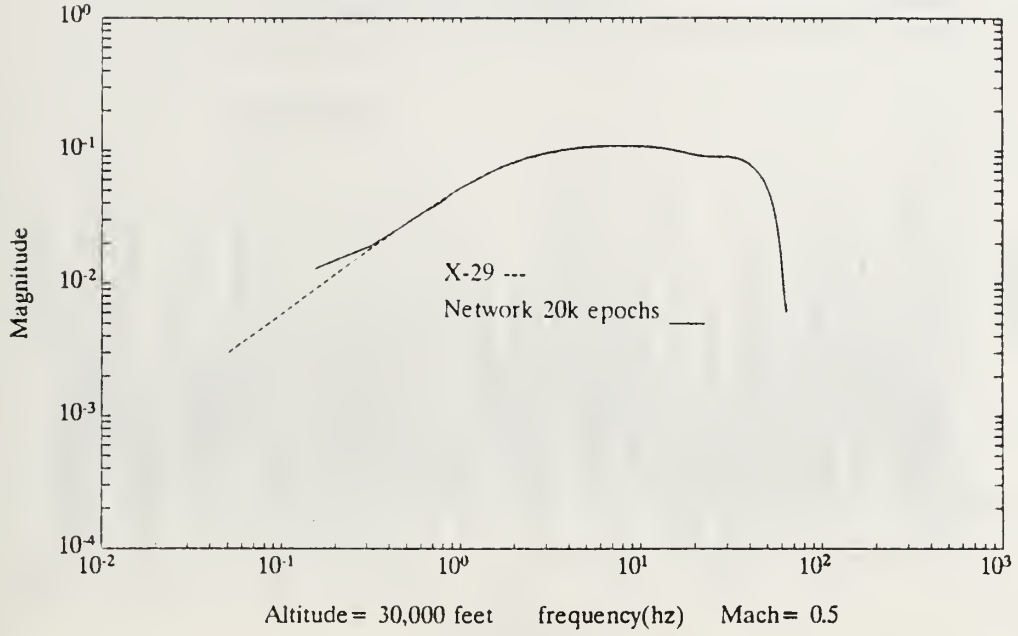


Figure 37 X-29 Model and Network α Frequency Responses to Input 2 (Optimal case)

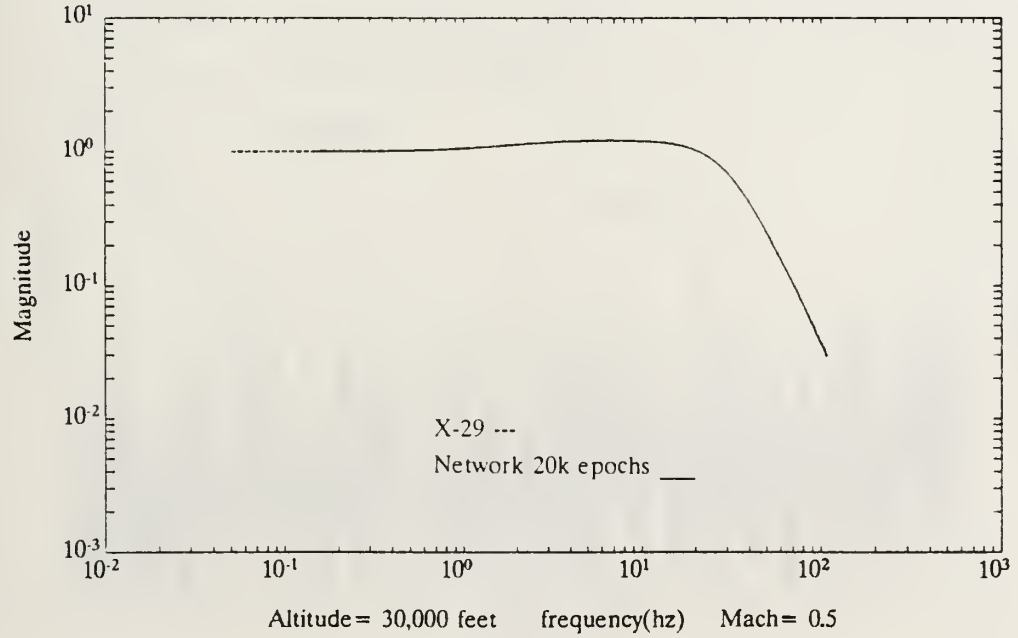


Figure 38 X-29 Model and Network q Frequency responses to Input 2 (Optimal case)

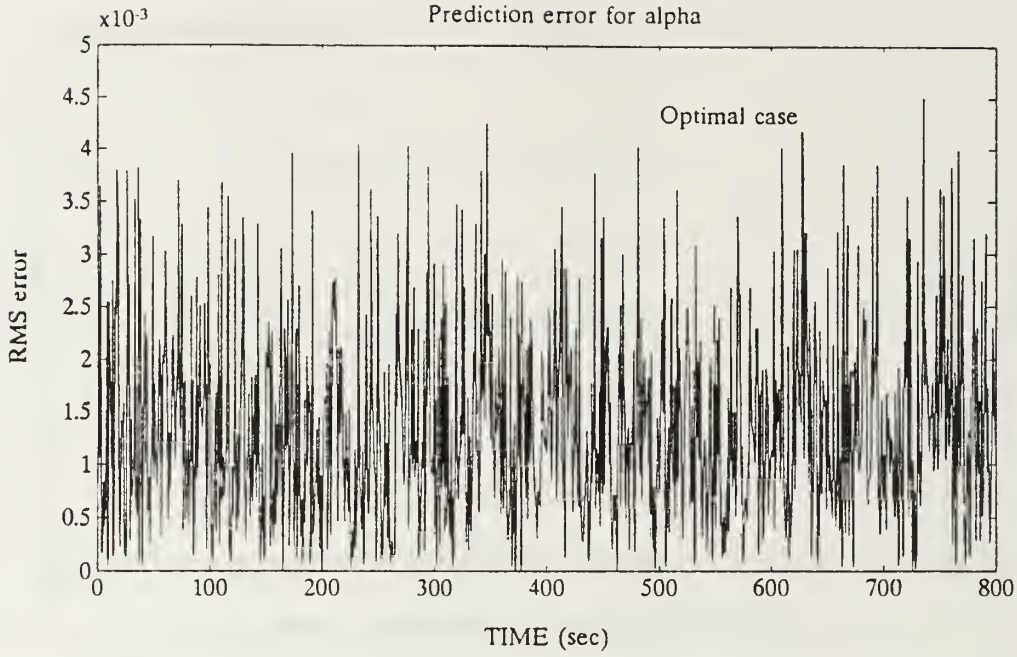


Figure 39 RMS Prediction Errors for α

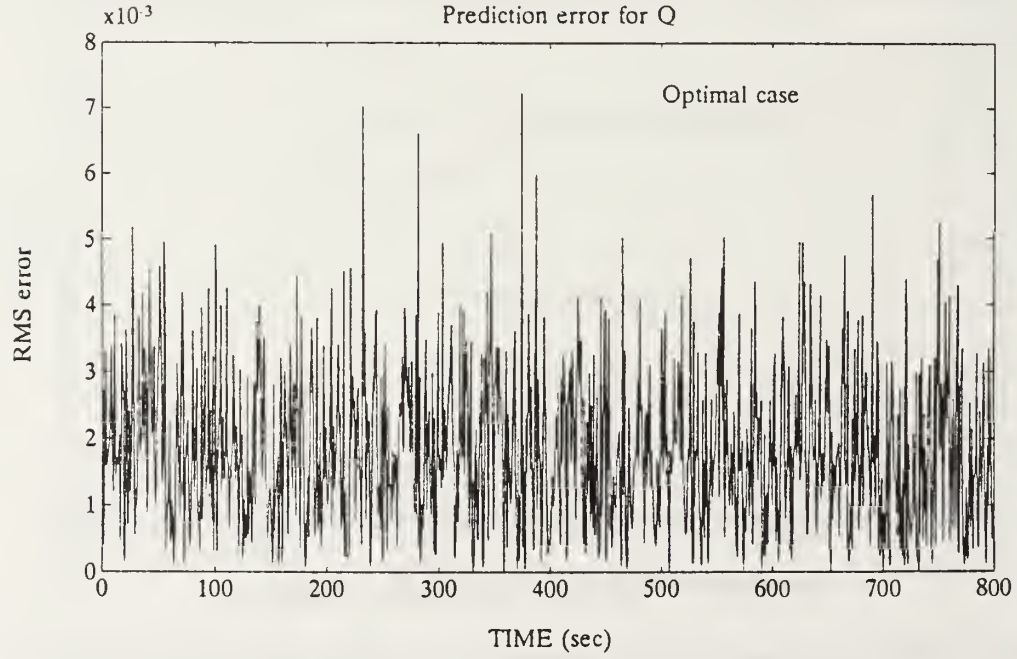


Figure 40 RMS Prediction Errors for q

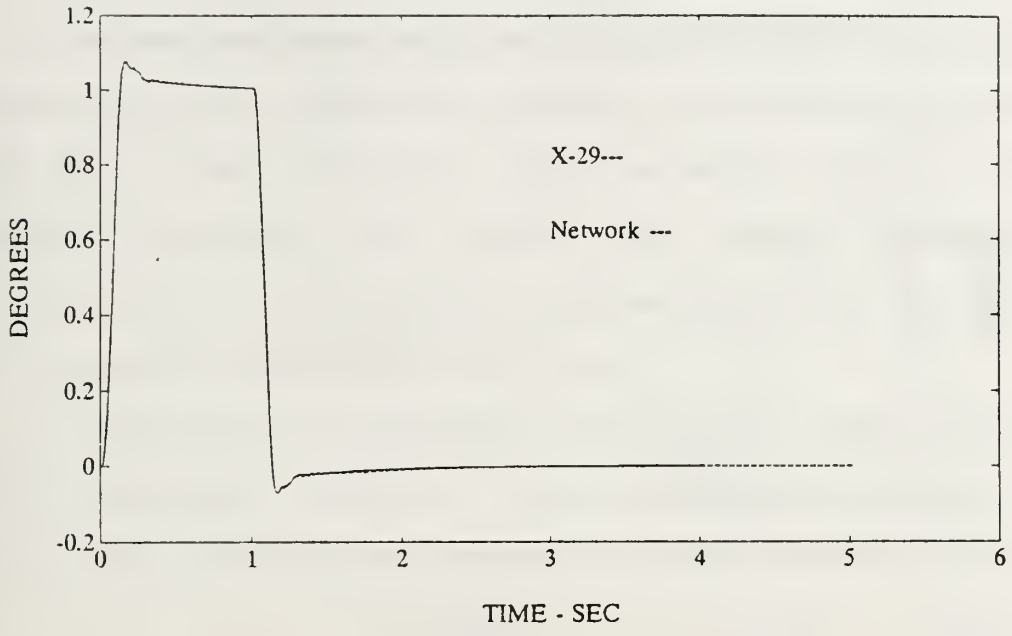


Figure 41 X-29 Model and Network α Time Responses to Input 1 (Optimal case)

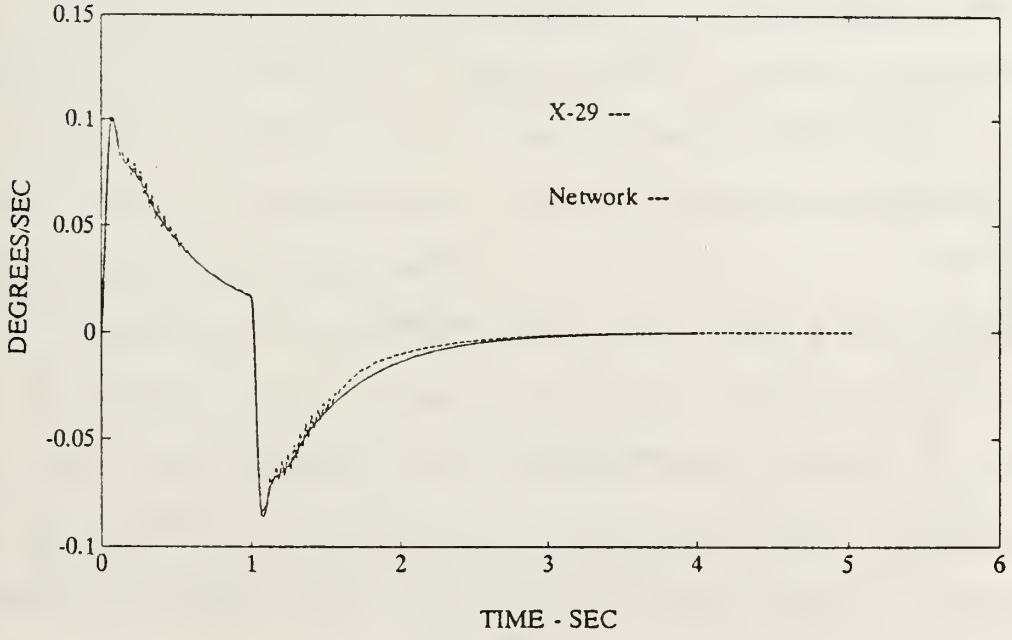


Figure 42 X-29 Model and Network q Time Responses to Input 1 (Optimal case)

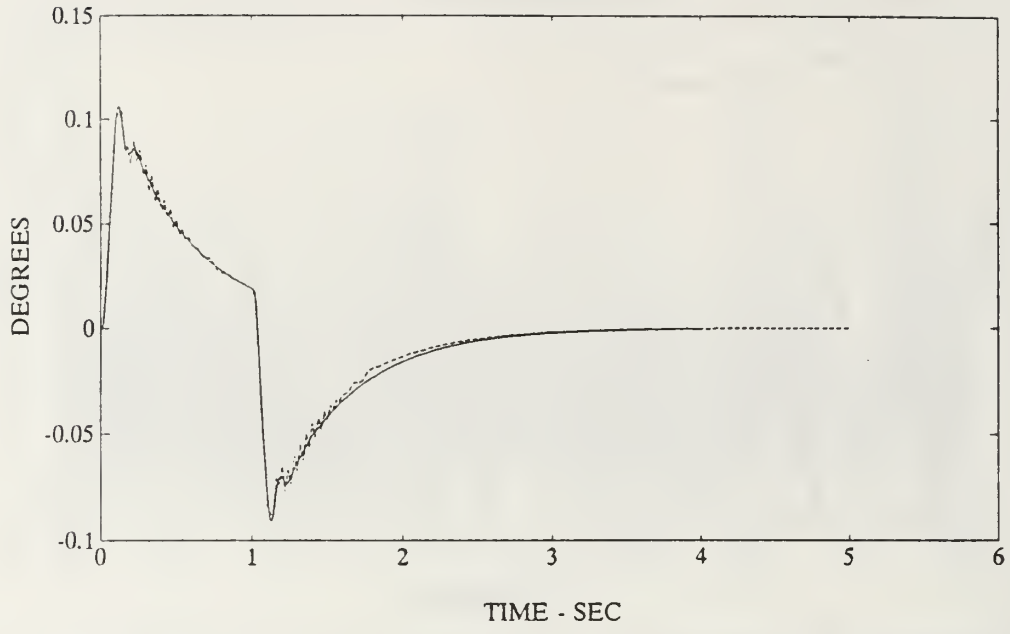


Figure 43 X-29 Model and Network α Time Responses to Input 2 (Optimal case)

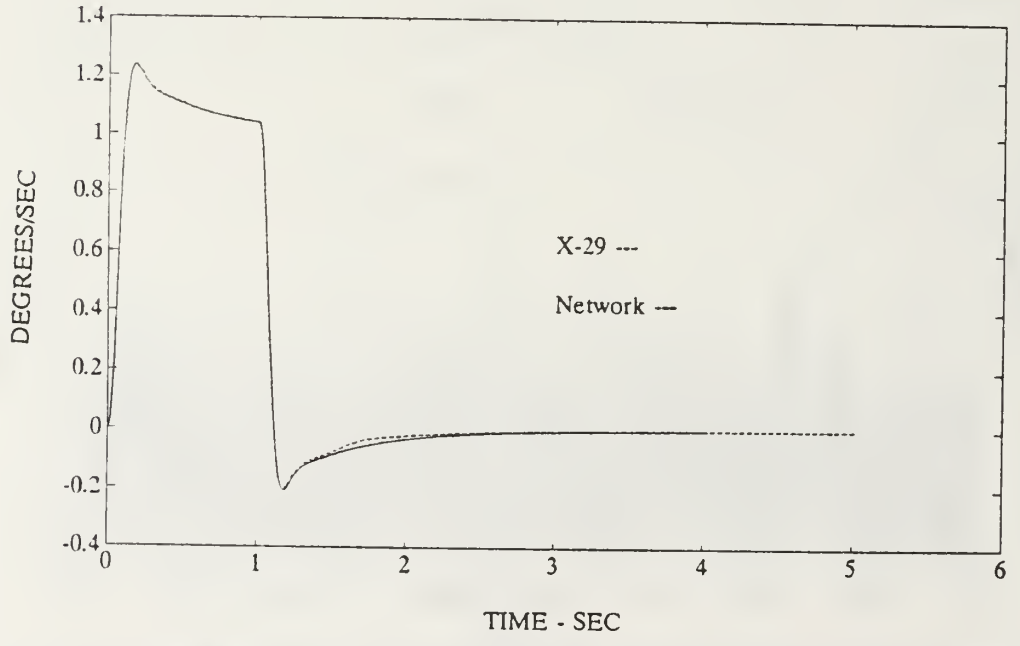


Figure 44 X-29 Model and Network q Time Responses to Input 2 (Optimal case)

2. Case #2 - Limited Performance X-29 Closed-Loop Plant

This second case emulates the limited performance X-29 closed-loop plant. The neural network is trained using also a single MIMO model structure fully connected with a linear activation function. The network has learned to respond correctly after 20,000 cycles or 300 seconds using two random binary inputs of magnitude 1.

The frequency responses of the linear neural network and the frequency responses of the limited performance X-29 model to two random binary inputs are displayed with discrete Bode plots in Fig. 45 through Fig. 48. The frequency responses of $\alpha(t)$ and $q(t)$ to input 1 and 2, which are shown in Fig. 45 through 48, develop near to exact model solutions. As with the optimal case #1, the frequency responses of $\alpha(t)$ to input 1 and 2 show unmodelled noise dynamics around the sampling frequency of 50 hertz, as shown in Fig. 45 and 47. Further training did not improve the present results. Contrary to the optimal case #1, the network is better able to model the low frequencies of both inputs.

RMS prediction error plots for α and q are given in Fig. 49 and 50. Notice that the vertical scales are on the order 10^{-3}. As expected, the network has learned very well with RMS errors on the order of 0.001 for α or 0.1 percent of the maximum output value of one and on the order of 0.002 for q.

The network has also learned to model correctly the limited performance X-29 closed-loop plant in the time domain. The X-29 model and the network time responses to input 1 and 2 are given in Fig. 51 through Fig. 54. As with the optimal performance X-29 of case #1, the limited performance X-29 model and the network responded to input 1 with a positive α and to input 2 with a semi-positive q, as shown in Fig. 51 and 54. However, the decoupling of α and q is not as pronounced as in the optimal performance case, i.e., this time, the $q(t)$ responses in Fig. 52 and the $\alpha(t)$ responses in Fig. 53 are not negligible.

The step responses of input 2 with an α rise time of 0.8 second, in Fig. 53, and with a q rise time of 0.5 second, in Fig. 54, indicate that the limited performance X-29 model and the network are slower to react than the optimal performance case #1. In the optimal case #1, the rise time of α to input 2 was 0.180 second and the rise time of q to input 2 was 0.095. The slower reaction times of the limited performance case is due to the fact that the control surface deflections and the control rates of the X-29 airplane were reduced to conform with the actuators limitations [Ref. 2].

In summary, the first configuration could simulate case #1, the large order and stable optimal performance X-29 closed-loop plant, and case #2, the large order and stable limited performance X-29 closed-loop plant, with high accuracy.

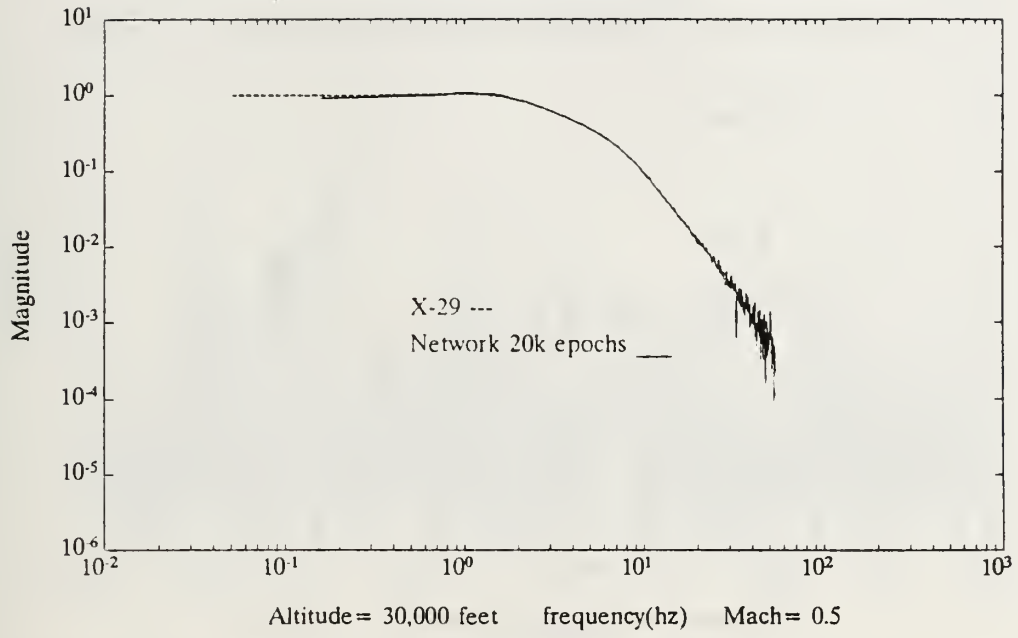


Figure 45 X-29 Model and Network α Frequency Responses to Input 1 (Limited case)

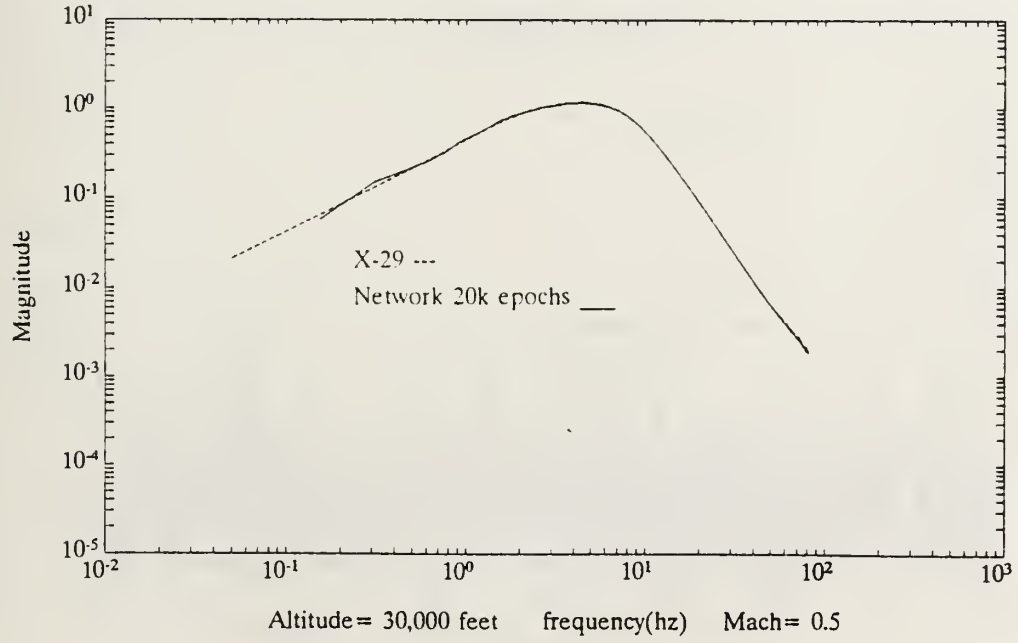


Figure 46 X-29 Model and Network q Frequency responses to Input 1 (Limited case)

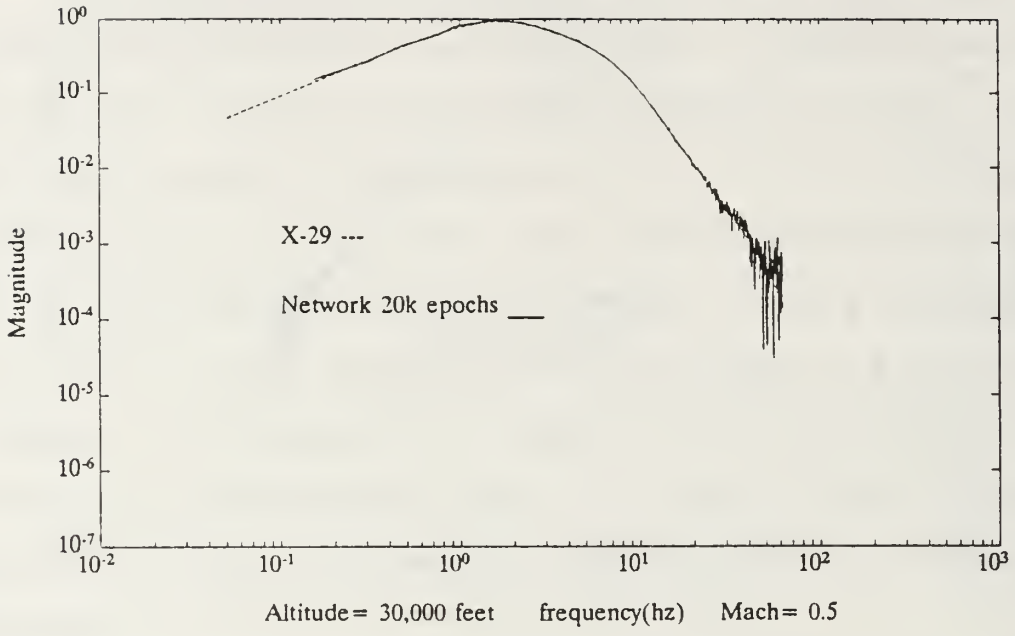


Figure 47 X-29 Model and Network α Frequency Responses to Input 2 (Limited case)

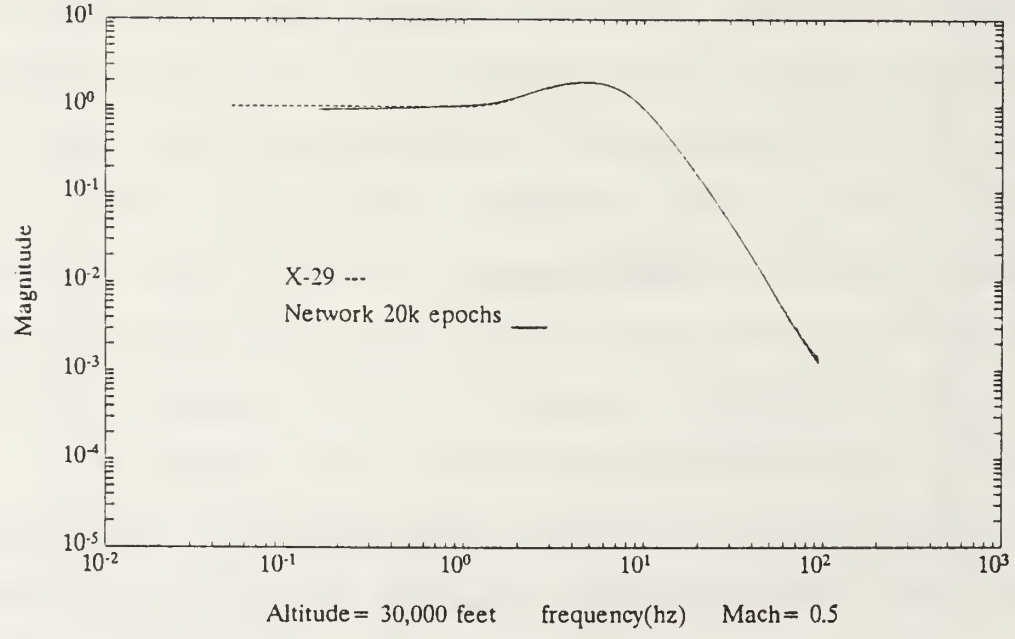


Figure 48 X-29 Model and Network q Frequency responses to Input 2 (Limited case)

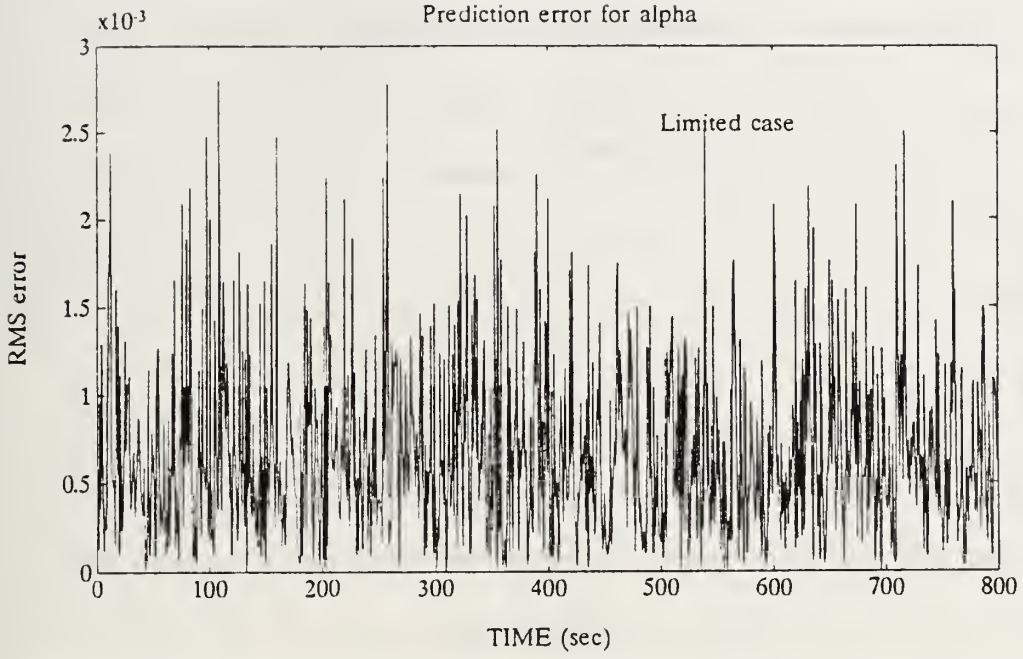


Figure 49 RMS Prediction Errors for α

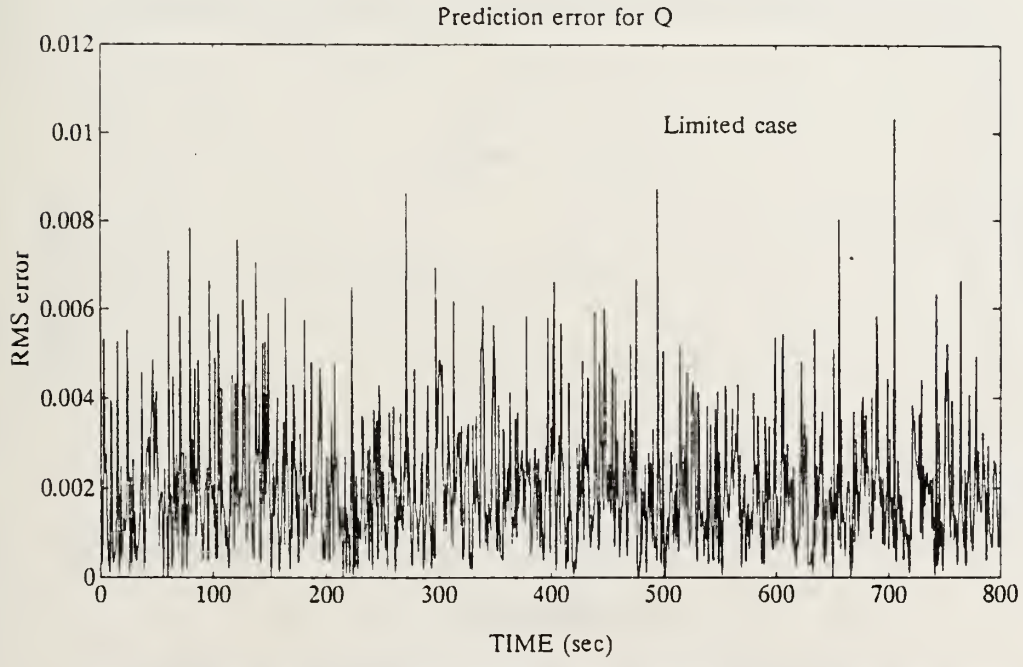


Figure 50 RMS Prediction Errors for q

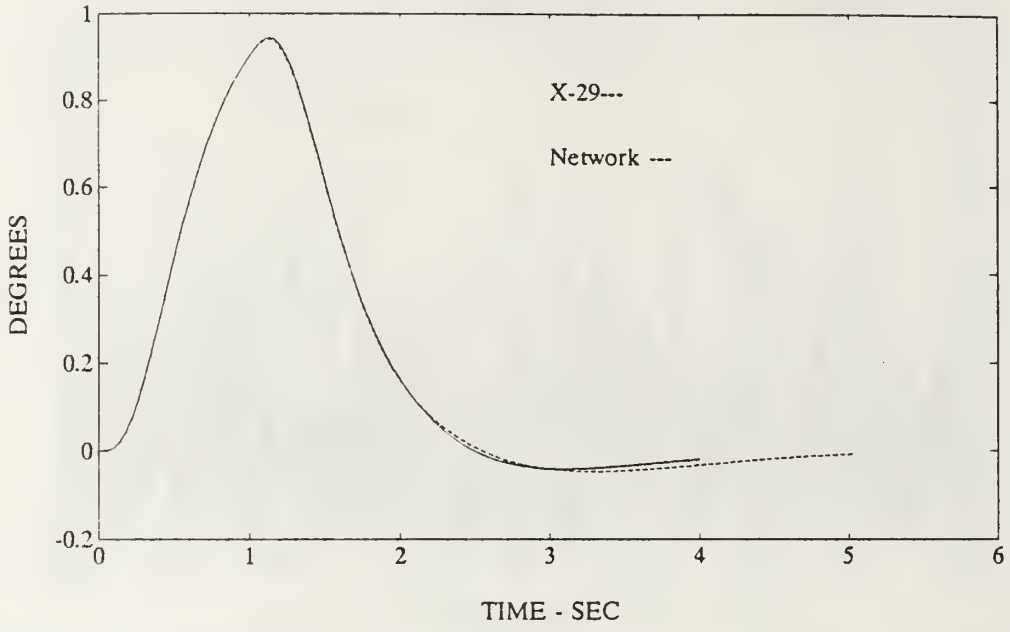


Figure 51 X-29 Model and Network α Time Responses to Input 1 (Limited case)

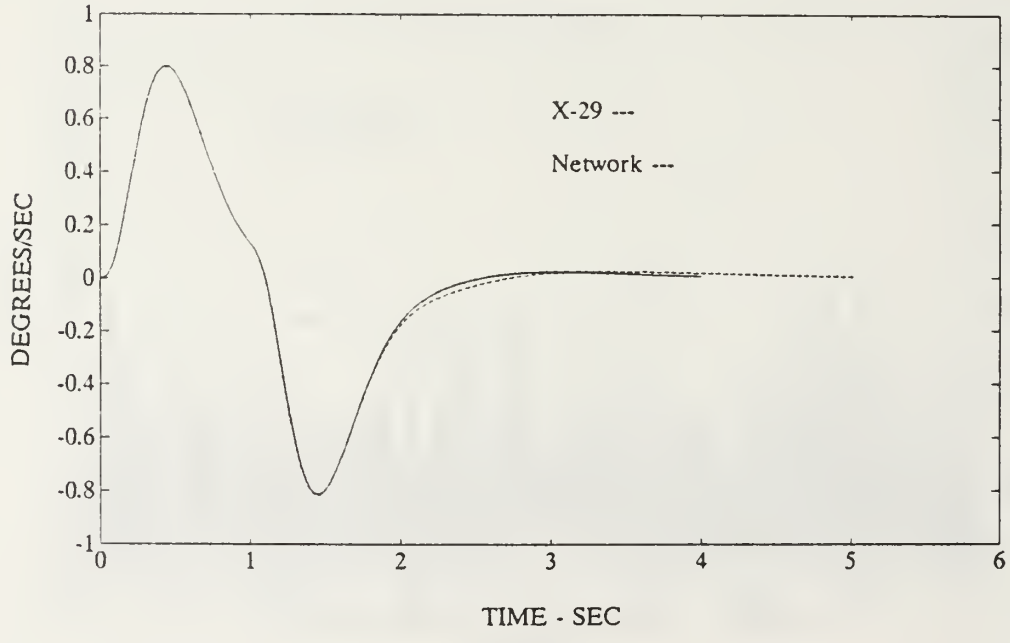


Figure 52 X-29 Model and Network q Time Responses to Input 1 (Limited case)

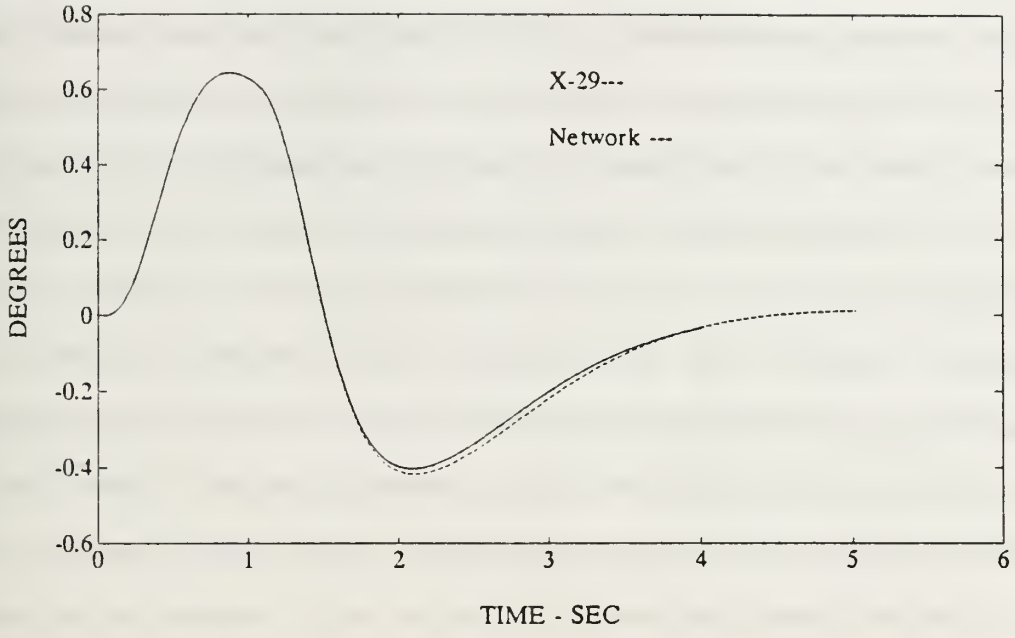


Figure 53 X-29 Model and Network α Time Responses to Input 2 (Limited case)

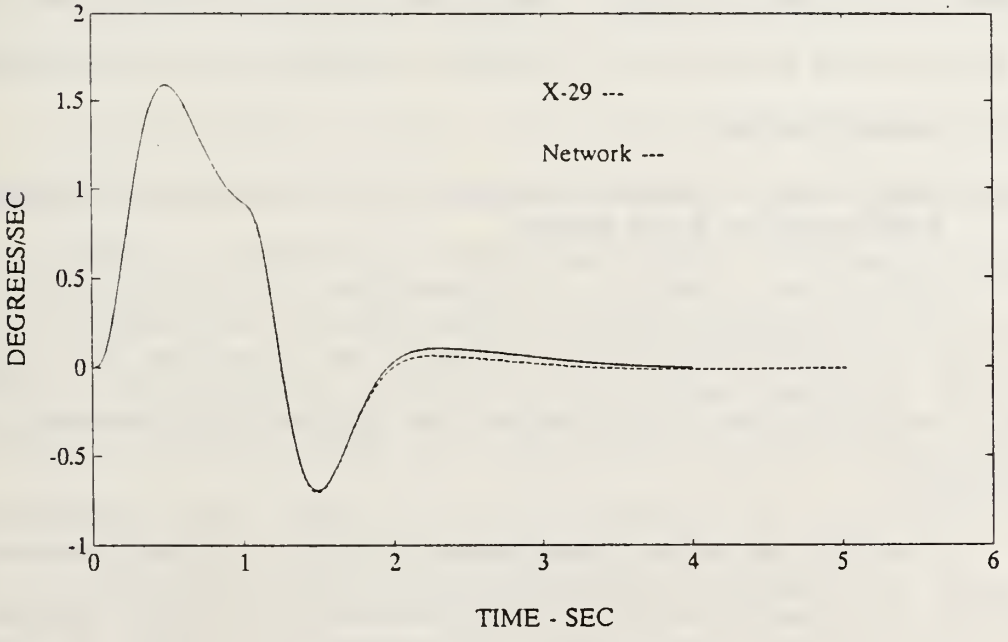


Figure 54 X-29 Model and Network q Time Responses to Input 2 (Limited case)

B. CONFIGURATION #2: IDENTIFICATION OF THE INVERSE PLANT

The second configuration is divided into three cases. Each case contains two levels of model structure as shown in the inverse plant architecture of Fig. 29 in Chapter V. The neural network representation of the inverse plant architecture is illustrated in Fig. 32. The SIMO neural network structure of the first level in Fig. 32 simulates the transfer function or the plant, and the MISO neural network structure of the second level in Fig. 32 simulates the inverse transfer function or the inverse plant. Before attempting to test the inverse plant neural network structures of Fig. 32 with unstable systems like the A-4D or the X-29 inverse plant, the stable inverse 30 states closed-loop transfer function of the optimal performance X-29 model will be examined first. Then, the testing of the small order, unstable inverse plant of the A-4D will follow and finally the unstable inverse plant of the X-29 will be investigated.

1. Case # 3 - Inverse Closed-Loop Plant of the Optimal Performance X-29 Model

Since the transfer functions for $\alpha(t)$ and $q(t)$ do not have any non-minimum phase zeros, the inverse 30 states closed-loop transfer functions of the optimal-performance case are stable.

The two networks' structure of Fig. 32 have learned to model the inverse transfer function within 450,000 epochs or 11,250 seconds. The SIMO network structure of Fig. 32

representing the 30 states closed-loop transfer function and the MISO network structure of Fig. 32 representing the inverse transfer function have been trained simultaneously. As with the first configuration, a random binary swept square wave of magnitude 1 was also the input signal at level 1.

After the two networks of Fig. 32 are fully trained, the random binary step outputs of the inverse transfer function at level 2 (output 2) should be equal to the random binary step input of the transfer function at level 1 (input 1). A time history, based on the number of epochs, of the comparison between the input of the 30 states closed-loop transfer function and the output of the inverse transfer function to a random binary input signal is shown in Fig. 55 through Fig. 62. The RMS errors between the two signals, output 2 and input 1, are also included in the list of figures.

As indicated in Fig. 55, after 2000 epochs or 500 seconds, the output of the inverse transfer function (output 2) is poorly correlated with the input of the transfer function (input 1). The RMS error is approximately 1.00 or 100 percent of the maximum output value of one, as shown in Fig. 56. After 150,000 epochs or 3750 sec, output 2 shows some similarities with input 1, as indicated in Fig. 57. The network emulating the inverse transfer function at level 2 learned to limit its output to values ± 1 and to follow the random binary step inputs of the transfer function at level 1

more closely. This time the RMS error, shown in Fig. 58, decreased from 1.0 to 0.35. After 250,000 epochs or 6250 sec, output 2 shows even better similarities with input 1, as demonstrated in Fig. 59. Only the magnitude of the steps need to be worked on. In this third trial, the RMS error in Fig. 60 is on the order of 0.175. Finally in Fig. 61, after 450,000 epochs or 11,250 sec, the output of the inverse transfer function (output 2) shows a near to exact solution to the input of the transfer function (input 1). The lowest RMS error obtained is on the order of .008 or 0.8 percent of the maximum output value of one, as shown in Fig. 62.

The discrete Bode plots in Fig. 63 and 64 give the α and q frequency responses to input 1 of the true optimal performance X-29 closed-loop transfer function and of network 2, which emulates the inverse transfer function. The frequency responses of network 1, emulating the transfer function, were given in the optimal performance case #1.

As explained in Appendix D, the Bode plots are obtained using the spectral transfer function. The spectral transfer function is calculated using an output vector and an input vector. When dealing with the neural network transfer function of level 1 in Fig. 32, the input vector is composed of the random binary step input signals at input 1 and the output vector is composed of the α and q time responses at output 1. However, when dealing with the neural network inverse transfer function of level 2 in Fig. 32, the input

vector is composed of the random binary step responses at output 2 and the output vector is composed of the α and q input signals at input 2.

The frequency responses are well modelled across the spectrum with the exception of a minor deviation in the low frequency region of the network 2 q response in Fig. 64.

In both time and frequency domain the SIMO and MIMO neural network structures of Fig. 32 have learned to model the inverse transfer function of a large order, stable system.

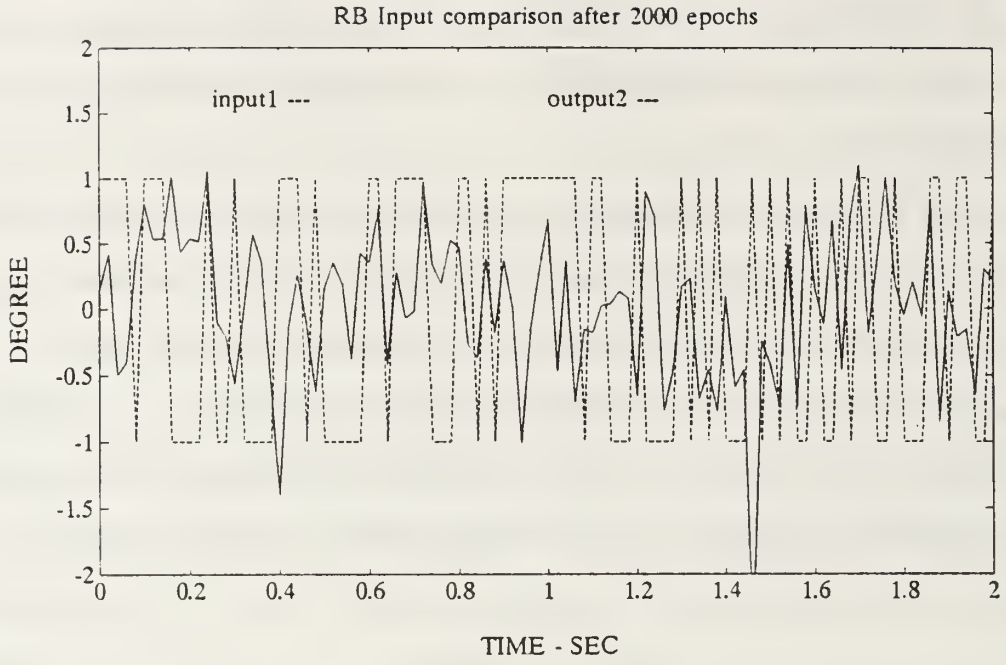


Figure 55 RB Input Comparisons after 2000 Epochs (case #3)

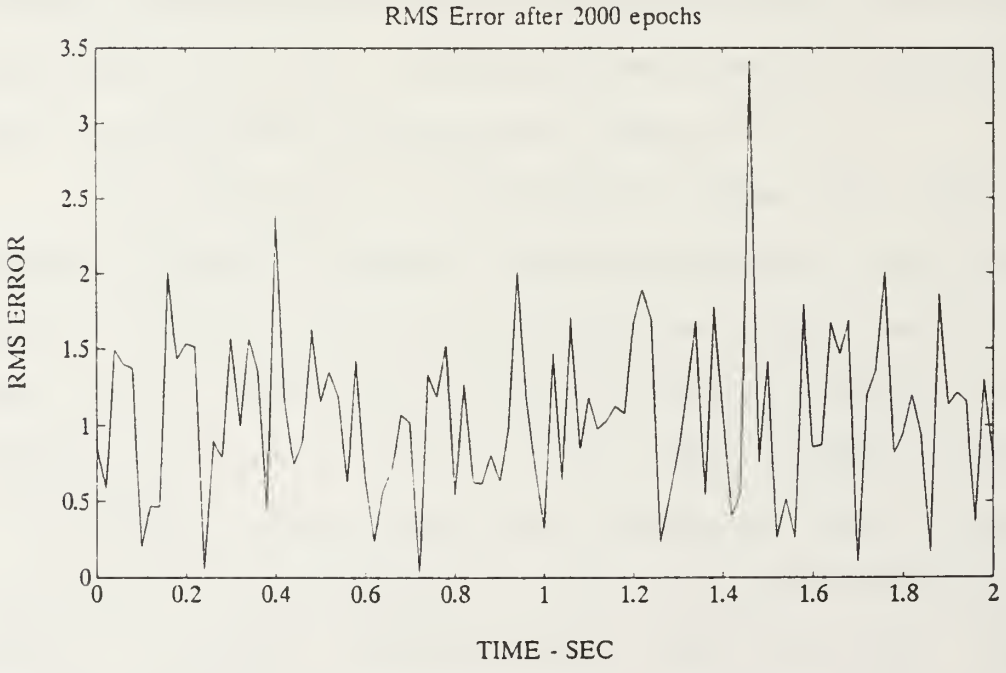


Figure 56 RMS Prediction Error after 2000 Epochs (case #3)

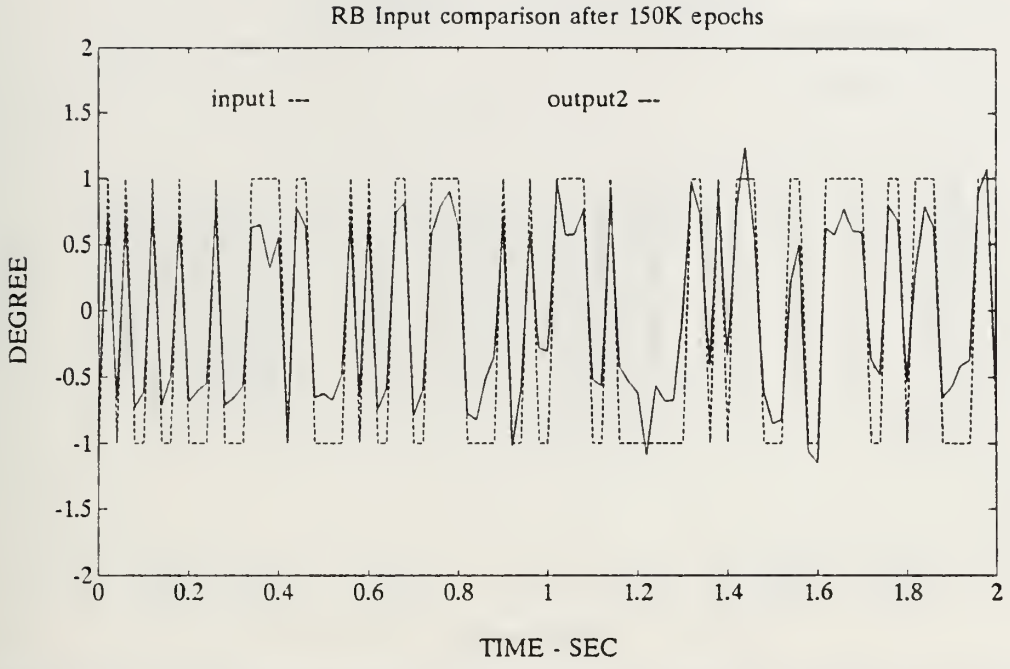


Figure 57 RB Input Comparisons after 150K Epochs (case #3)

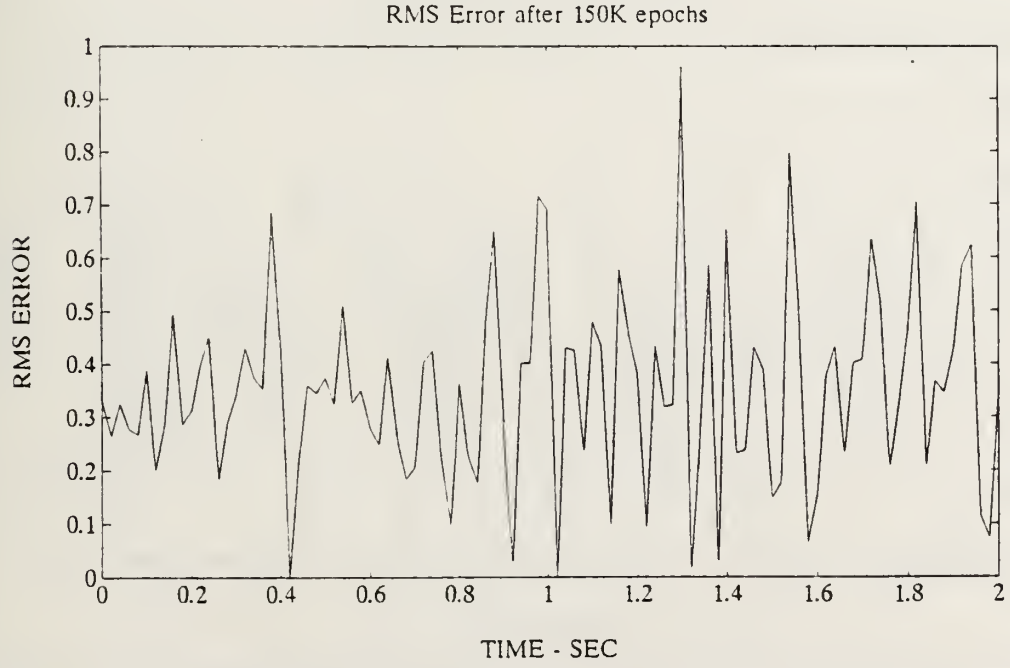


Figure 58 RMS Prediction Error after 150k Epochs (case #3)

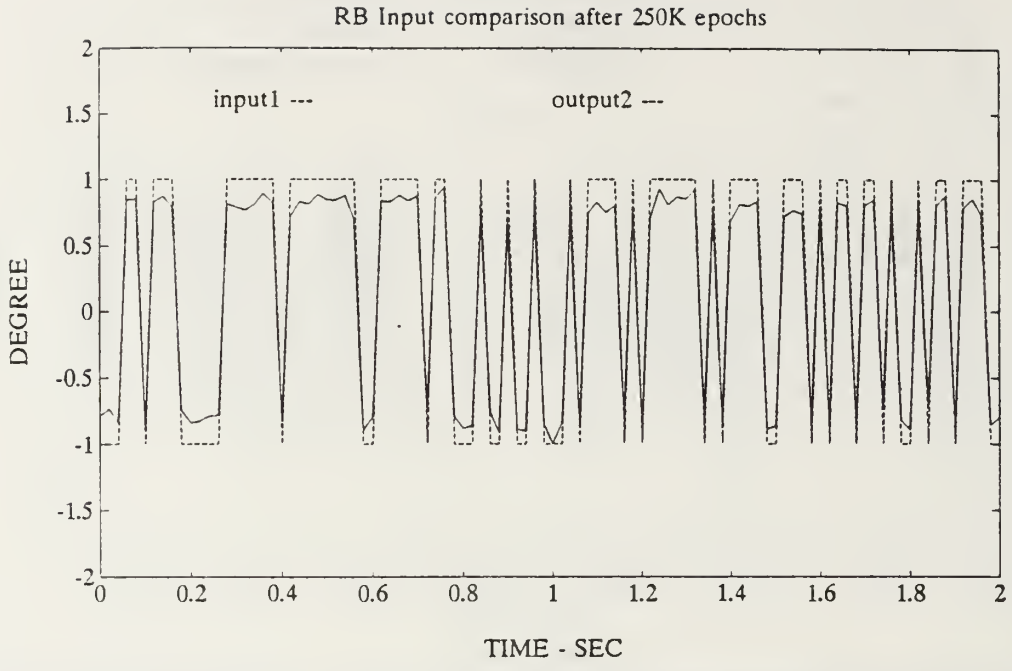


Figure 59 RB Input Comparisons after 250K Epochs (case #3)

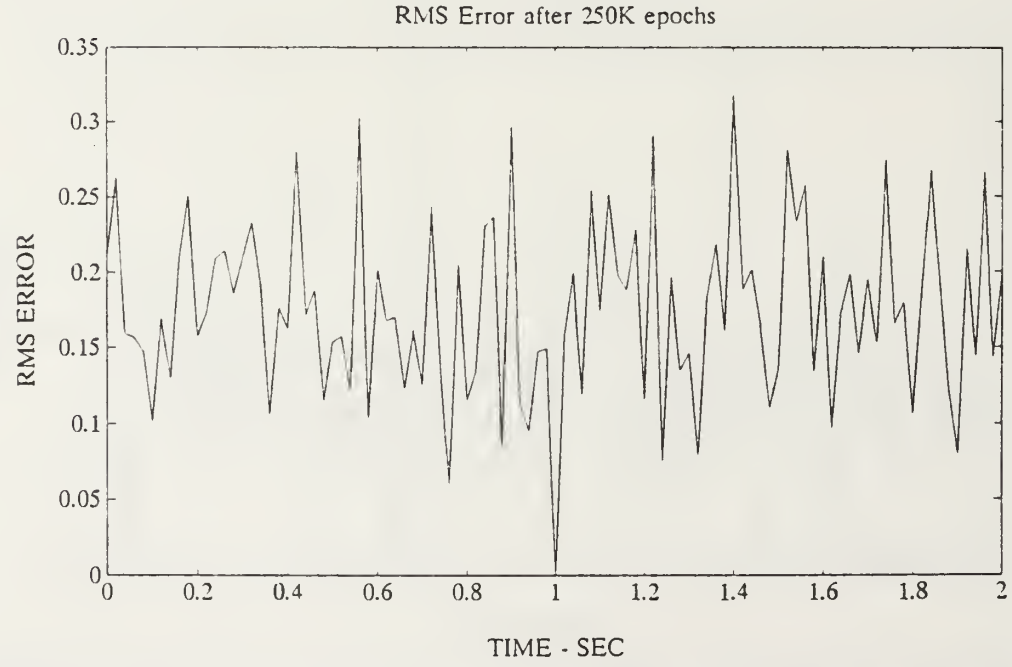


Figure 60 RMS Prediction Error after 250k Epochs (case #3)

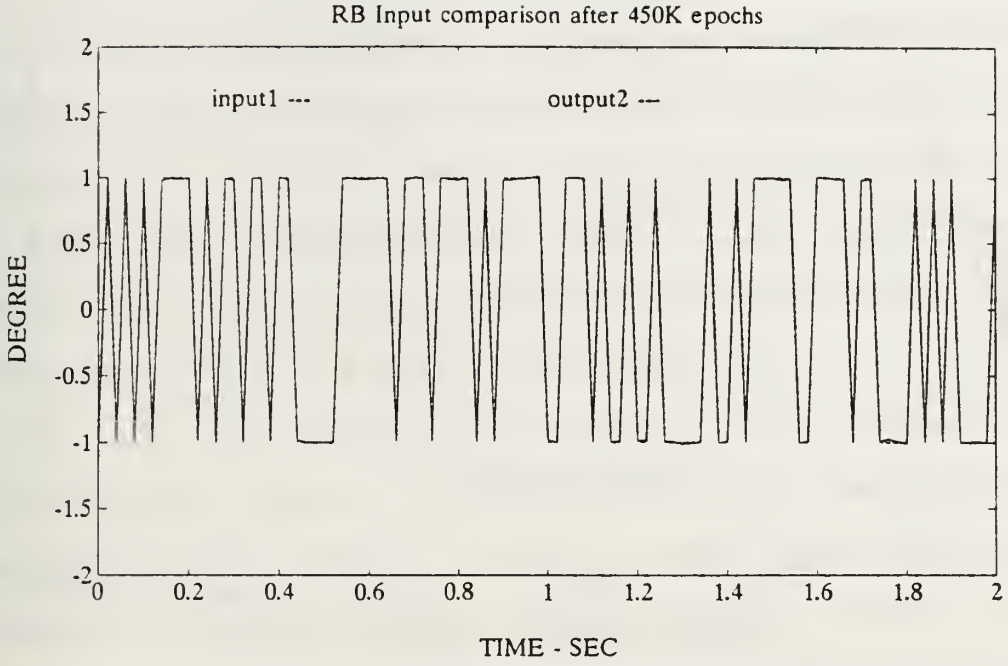


Figure 61 RB Input Comparisons after 450K Epochs (case #3)

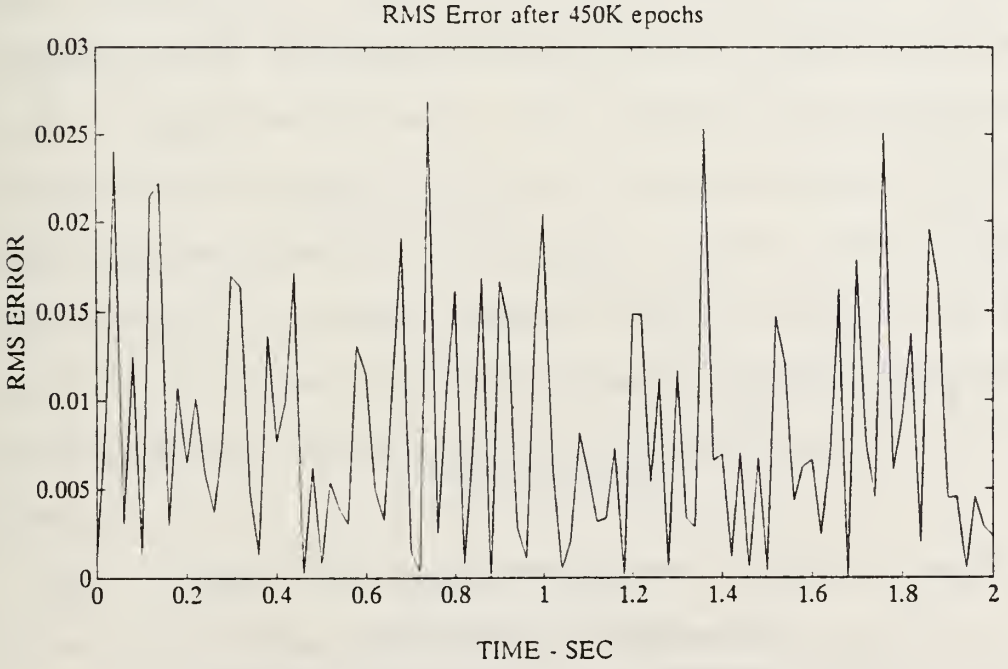


Figure 62 RMS Prediction Error after 450k Epochs (case #3)

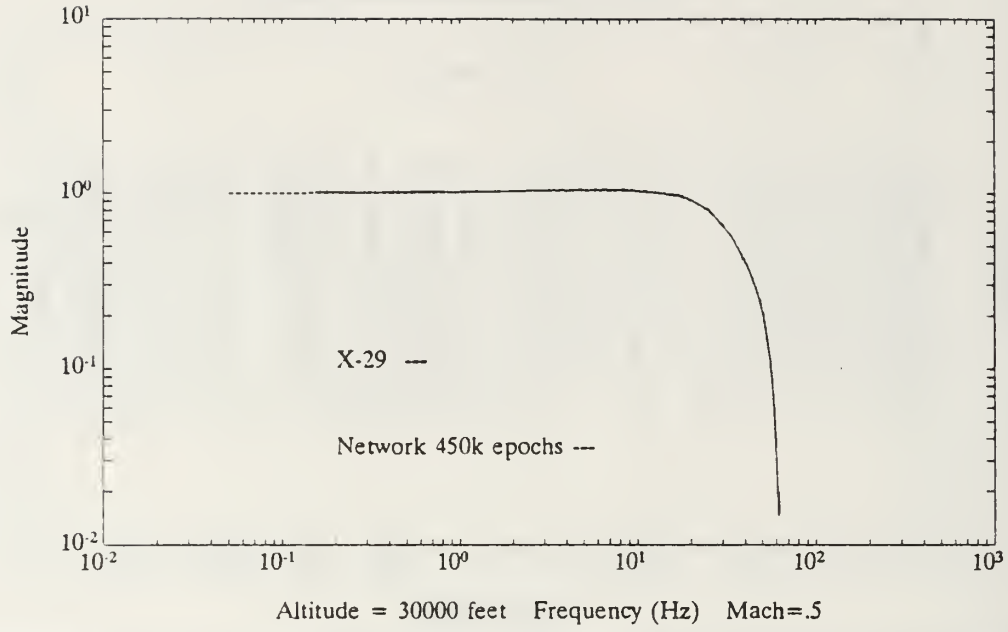


Figure 63 X-29 Inverse Plant Model and Network 2 α Frequency Responses to Input 1 (Optimal case)

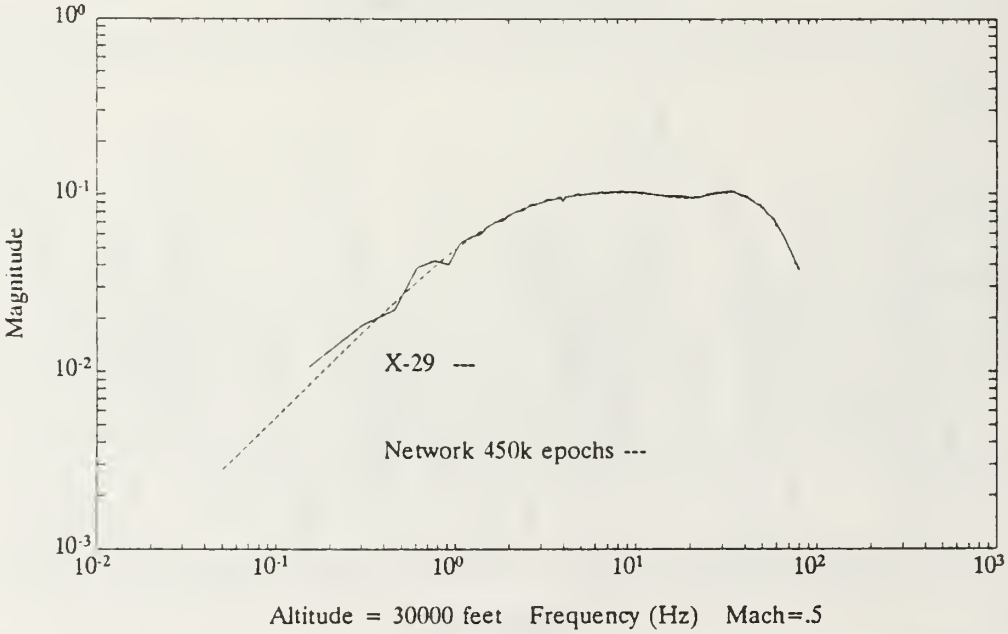


Figure 64 X-29 Inverse Plant Model and Network 2 q Frequency Responses to Input 1 (Optimal case)

2. Case # 4 - Inverse Plant of the A-4D Aircraft

The fourth case analyses an open-loop transfer function, the A-4D fourth order plant, instead of a closed-loop transfer function, the X-29 30th order system of case #3. As mentioned in the previous chapter, the A-4D plant is a small order system whose inverse is unstable due to the fact that the plant has a non-minimum phase zero.

In this configuration, level 1 of the inverse plant architecture of Fig. 29 represents the A-4D plant and level 2 represents its inverse. In Fig. 32, the same SIMO and MISO neural network structures of the X-29 case #3 were used for this case with the exception of three modifications. The first modification is that there are four outputs to the A-4D plant at level 1 ($u,\alpha,q,$ *and theta*) instead of two for the X-29 plant (α *and* q) , thus there are four inputs to level 2 instead of two. The second modification is that the network of level 2 emulates the small and unstable A-4D inverse plant instead of the large and stable X-29 inverse plant. Finally, the third modification is that the sampling time for the A-4D aircraft is 0.1 seconds instead of 0.02 seconds for the X-29 aircraft. The sampling time being 0.1 seconds indicates that the A-4D longitudinal modes are slower than the X-29 dynamic modes.

The SIMO and the MISO networks of Fig. 32 have learned to model the A-4D inverse plant after 25,000 cycles or 450 seconds. The random binary step outputs of the inverse plant

network 2 (output 2) shows a very near to exact solution to the random binary step inputs of the plant network 1 (input 1), as shown in Fig. 65. The lowest RMS error obtained is on the order of 0.00015 or 0.015 percent of the maximum output value of one, as indicated in Fig. 66. Notice that the vertical scales are on the order 10^{-4}.

As expected, the speed of the simulation was much faster for the A-4D model of case #4 than for the X-29 model of case #3 since the order of the system is seven and a half times smaller. The order of the system in addition to the number of inputs and outputs determines the number of elements in the regression vector. The number of elements in the regression vector determines the number of connections in the neural network structure, which in turn determines the speed of the simulation. The smaller the order of the system, the smaller the number of connections required in the neural network, and therefore the faster the simulation.

The discrete Bode plots of Fig. 67 through Fig. 70 give the u, α, q, and *theta* frequency responses of the A-4D plant model and of network 1, which emulates the plant. The discrete Bode plots of Fig. 71 through Fig. 74 give the u, α, q and *theta* frequency responses of the A-4D inverse plant model and of network 2, which emulates the inverse plant.

As with the X-29 model case #3, the spectral transfer function of the A-4D inverse plant of level 2 has been calculated using the random binary step responses of output 2

as the input vector and the *u*, α, *q* and *theta* input signals of input 2 as the output vector. Both networks, network 1 simulating the A-4D plant and network 2 simulating the A-4D inverse plant, as shown in Fig. 67 through Fig. 74, have no difficulties to model the high frequency region. However, even with further training, the networks are unable to better model the low frequency region. The same results were obtained, in respect to the plant, by R. Scott [Ref. 1].

Knowing that the time responses of $\alpha(t)$ and $q(t)$ are predominantly of high frequency or short period mode, it can be seen looking from the Bode plots that the emulations of the A-4D plant and its inverse have been accomplished with high accuracy.

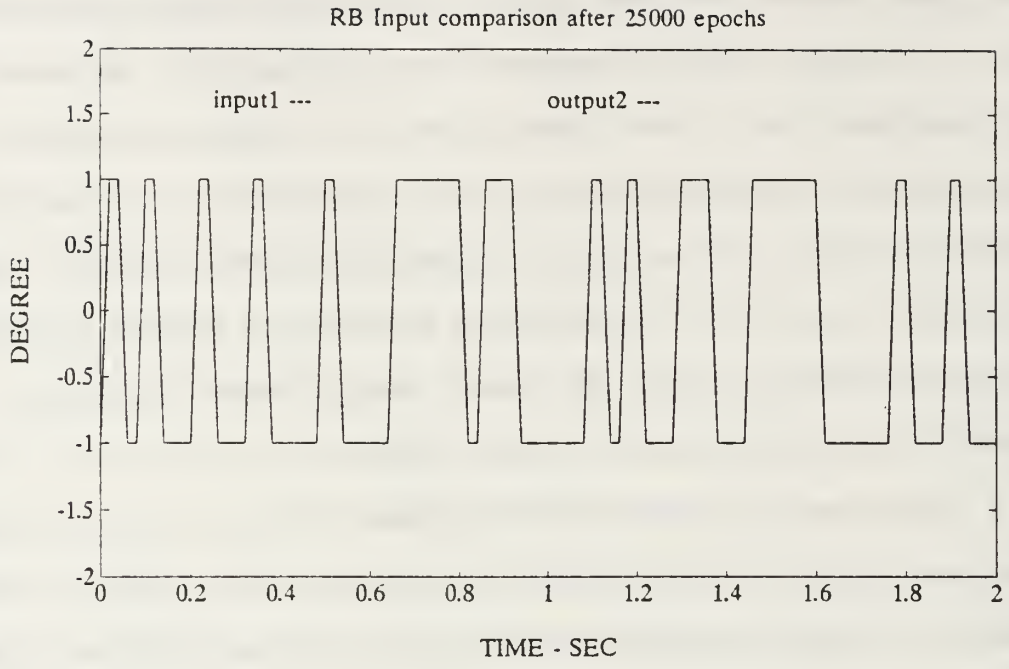


Figure 65 RB Input Comparisons after 25K Epochs (case #4)

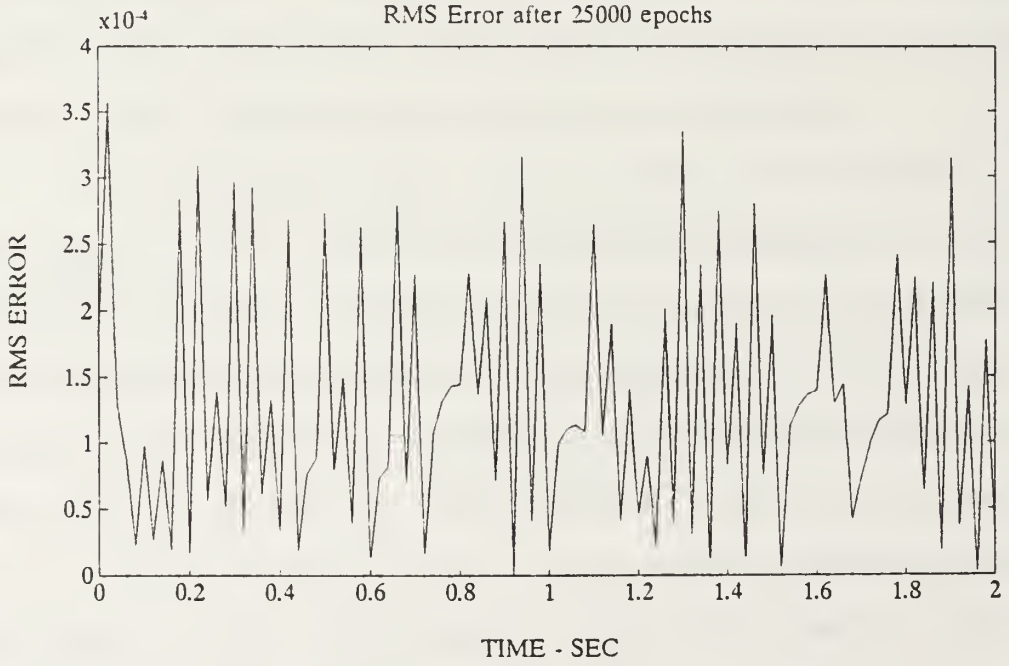


Figure 66 RMS Prediction Error after 25K Epochs (case #4)

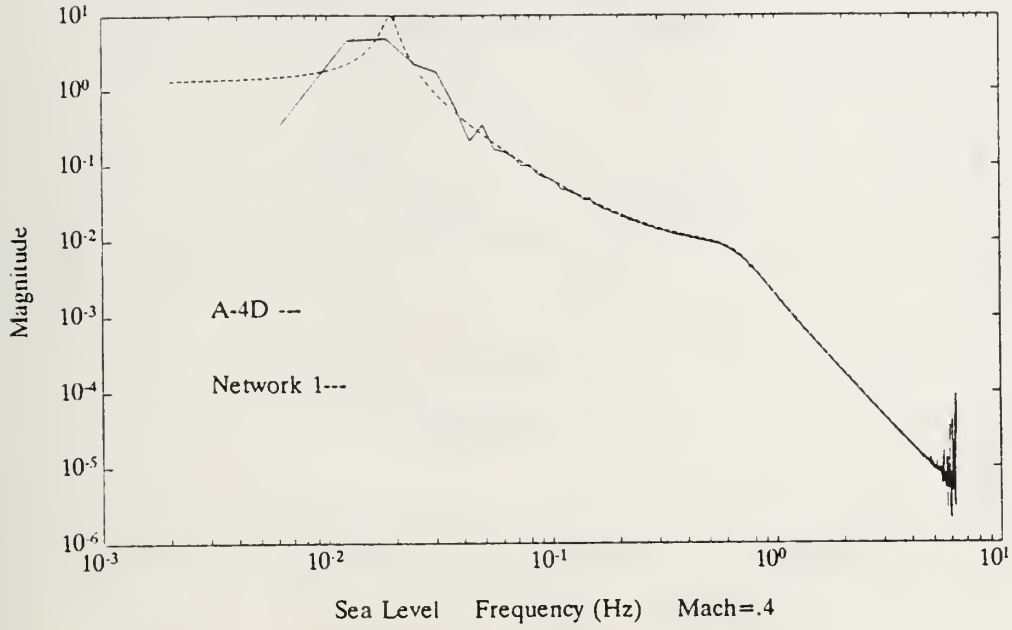


Figure 67 A-4D Plant Model and Network 1 u Frequency Responses

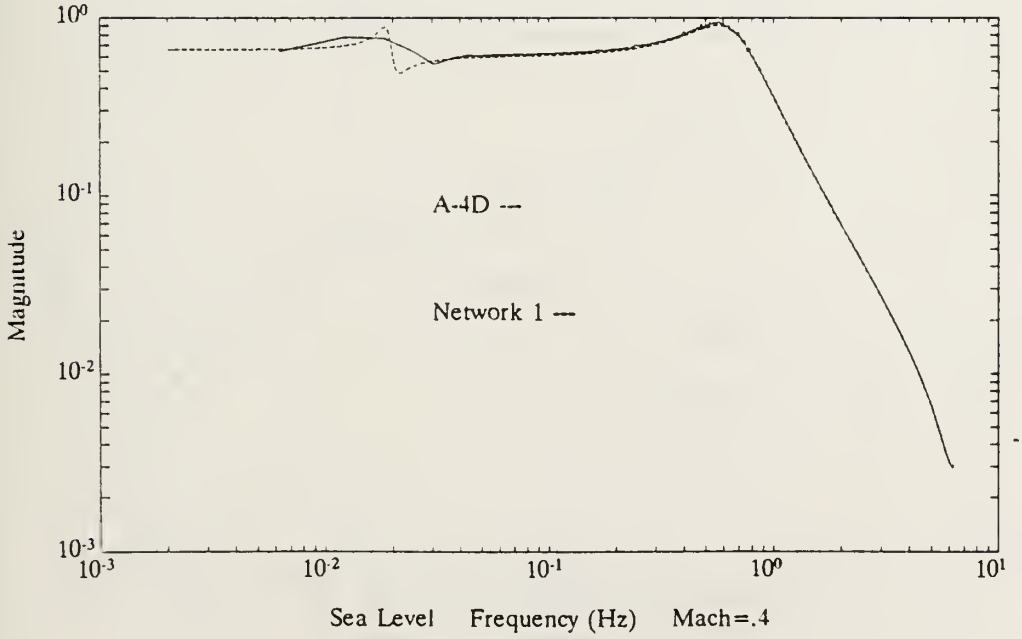


Figure 68 A-4D Plant Model and Network 1 α Frequency Responses

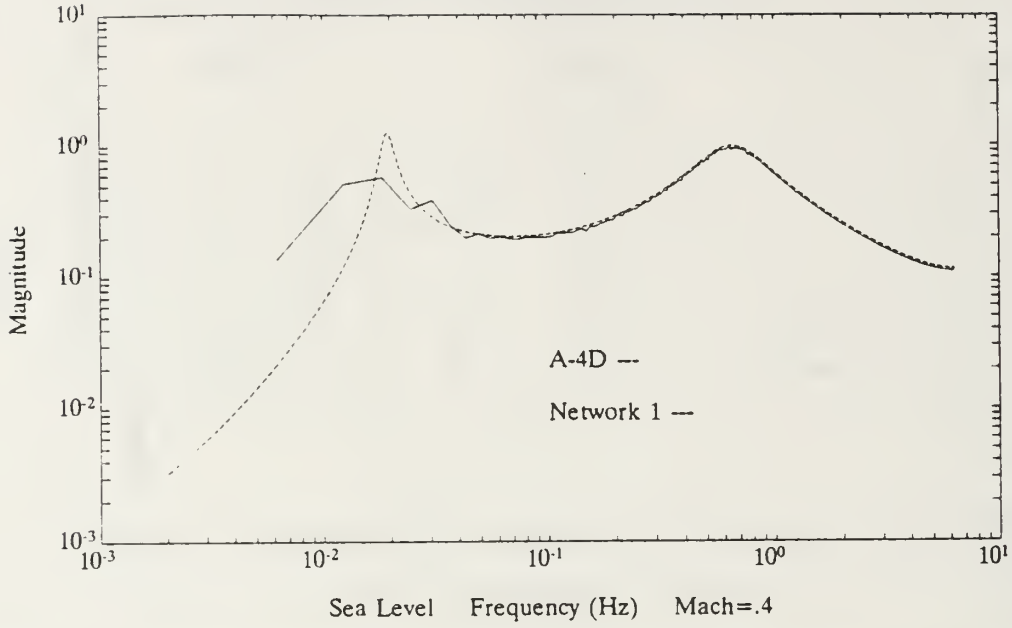


Figure 69 A-4D Plant Model and Network 1 q Frequency Responses

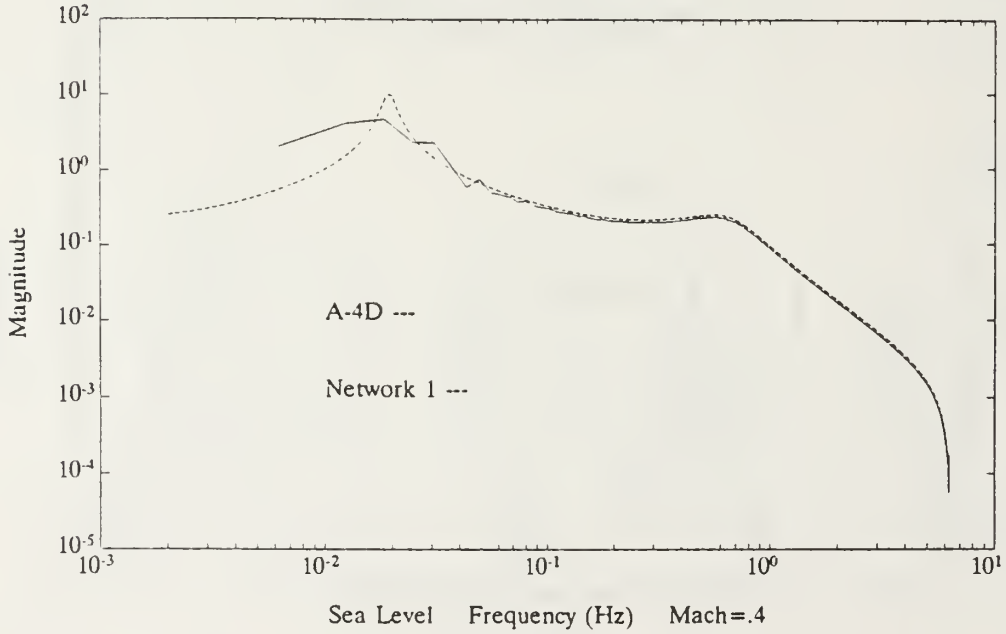


Figure 70 A-4D Plant Model and Network 1 theta Frequency Responses

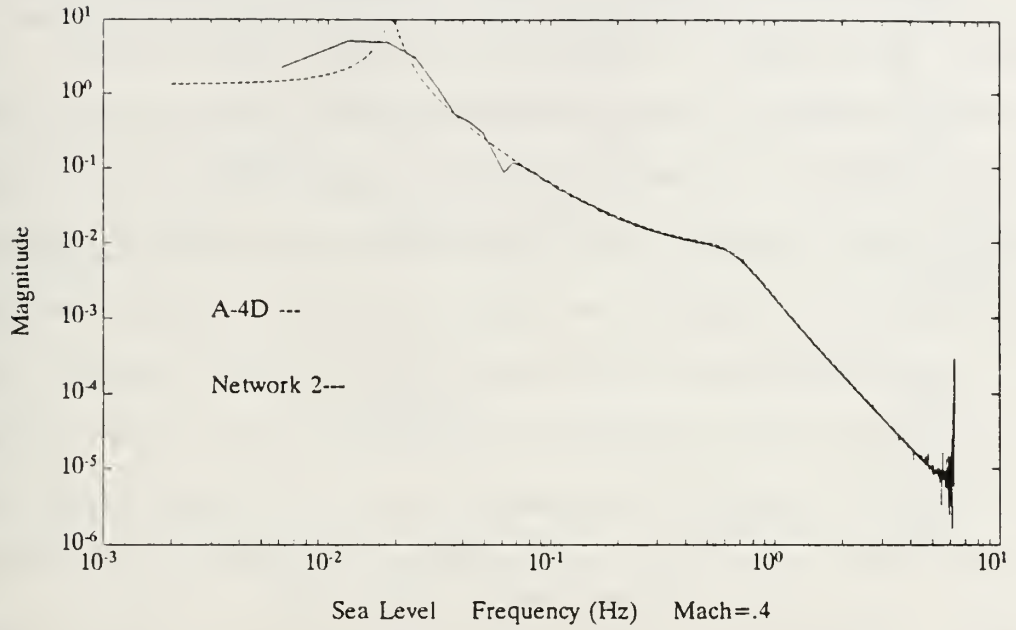


Figure 71 A-4D Inverse Plant Model and Network 2 u Frequency Responses

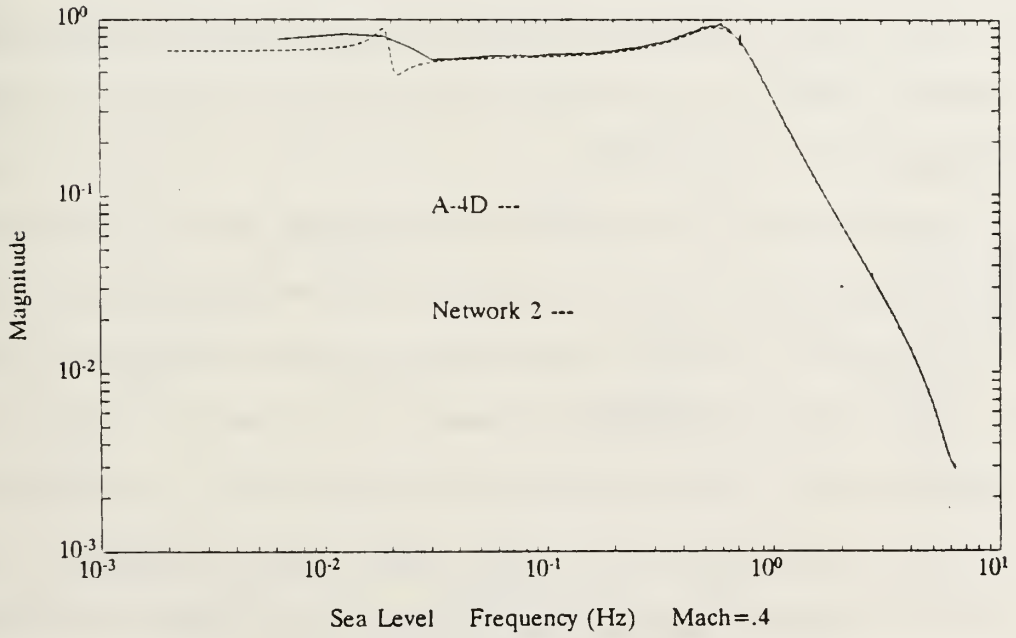


Figure 72 A-4D Inverse Plant Model and Network 2 α Frequency Responses

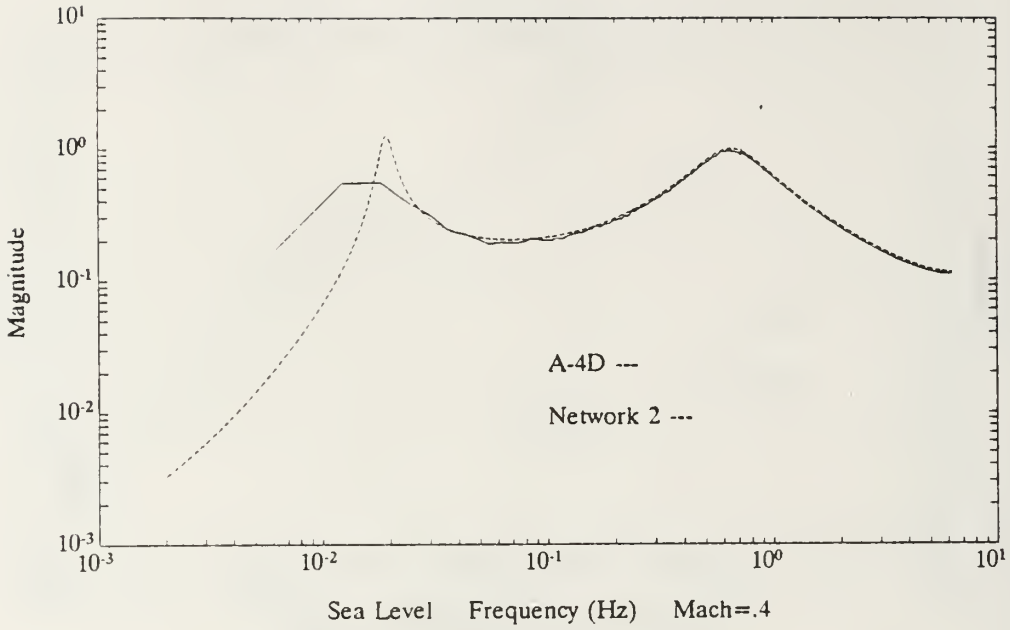


Figure 73 A-4D Inverse Plant Model and Network 2 q Frequency Responses

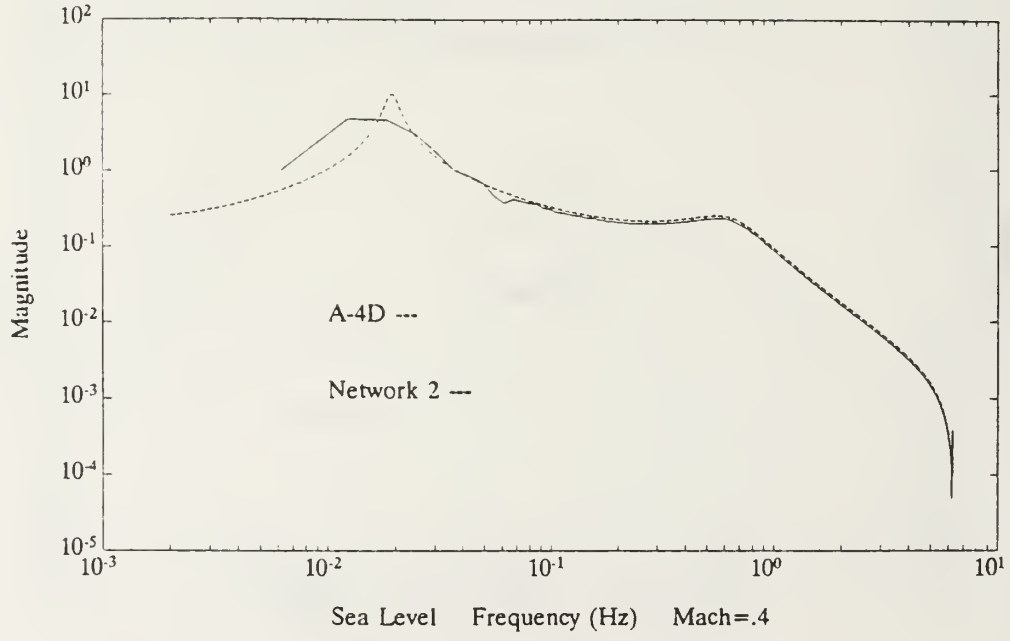


Figure 74 A-4D Inverse Plant Model and Network 2 theta Frequency Responses

3. Case # 5 - Inverse Plant of the X-29 Aircraft

The fifth case also analyses an open-loop transfer function, the X-29 fourteen order plant. As shown in Fig. 30 and 31 of Chapter V, the A-4D plant has no unstable pole and one non-minimum phase zero at -3.65, whereas the X-29 plant has one unstable pole at +1.05 and three non-minimum phase zeros located at -4.5, +2.2, and +9.5. As a consequence of the unstable pole, the X-29 plant is unstable. As a consequence of the size of the X-29 non-minimum phase zeros, +9.5 and -4.5, versus the A-4D non-minimum phase zero, -3.65, the degree of instability of the X-29 inverse plant is much higher than the degree of instability of the A-4D inverse plant.

In the inverse plant architecture of Fig. 29, the X-29 plant without controller, which has not been modelled prior to this case, is emulated with neural network 1. The X-29 inverse plant is emulated with neural network 2. As shown in Fig. 29, only one input to the plant is necessary to investigate this case study. Both inputs to the plant will be analyzed in the third configuration when simulating the controllers and the plant. The same SIMO and MISO neural network structures of the X-29 model of case #3 are used in addition to the same number of outputs, 2, and the same sampling time, 0.02 seconds. The only exceptions are that the neural network of level 1 emulates a smaller, unstable system (X-29 plant) and that the neural network of level 2 emulates a very unstable system (X-29 inverse plant).

After 25,000 epochs or 300 seconds, neural network 1 has learned to model the unstable X-29 plant in the time domain. The α and q time responses of the X-29 plant model and of the network 1 to a step of 0.01 degree for one second applied to input 1 are given in Fig. 75 and 76. The RMS errors are on the order of 0.0005 for α or 5.0 percent of the maximum output value of one, and on the order of 0.0001 for q. Notice in both figures the positive exponential departures of both responses with respect to time. This explains why a step input of 0.01 degree was chosen over the step input of one degree to test the system. These exponential responses can be controlled by limiting the plant outputs to a certain value. Limiting the plant outputs has two purposes. First, it avoids the plant output signals to grow exponentially. Second, since the plant outputs become the inverse plant inputs, it limits the control inputs of the inverse plant network 2. In this way the effects of the unstable inverse plant are restrained.

In case #5 of the USERIO program of Appendix A, the generation of the system or plant outputs are limited to values between $\pm$ 1. Every time one of the two plant outputs reaches $\pm$1, the same output is reset to zero. These resets occur approximately every 90 epochs or 1.8 seconds, which introduces noise every 0.5 hertz in the frequency spectrum, as demonstrated in Fig. 77 and 78. The mean of the noise dynamics in Fig. 78 follows the true plant response to a certain point, around 20 hertz. By applying an adequate filter, a proper

frequency response of the network could be more or less obtained.

The very unstable inverse plant makes the task of the inverse plant network 2 more difficult. Even after 500,000 epochs or 8,500 seconds, neural network 2 cannot emulate the inverse plant, as shown in Fig. 79. The RMS error is on the order of 0.6 or 60 percent of the maximum output of one.

In summary, the second configuration could simulate case #3, the large order and stable optimal performance X-29 inverse closed-loop transfer function and case #4, the small order, unstable A-4D inverse plant, but it could not simulate case #5, the more unstable X-29 inverse plant. The unstable X-29 plant was however emulated.

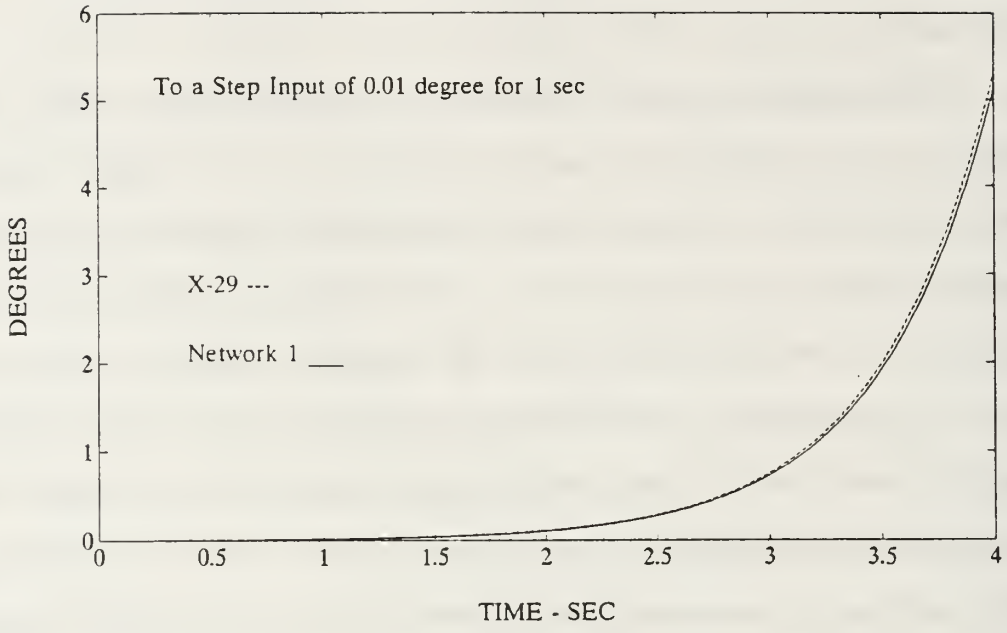


Figure 75 X-29 Plant Model and Network 1 α Time Responses to Input 1

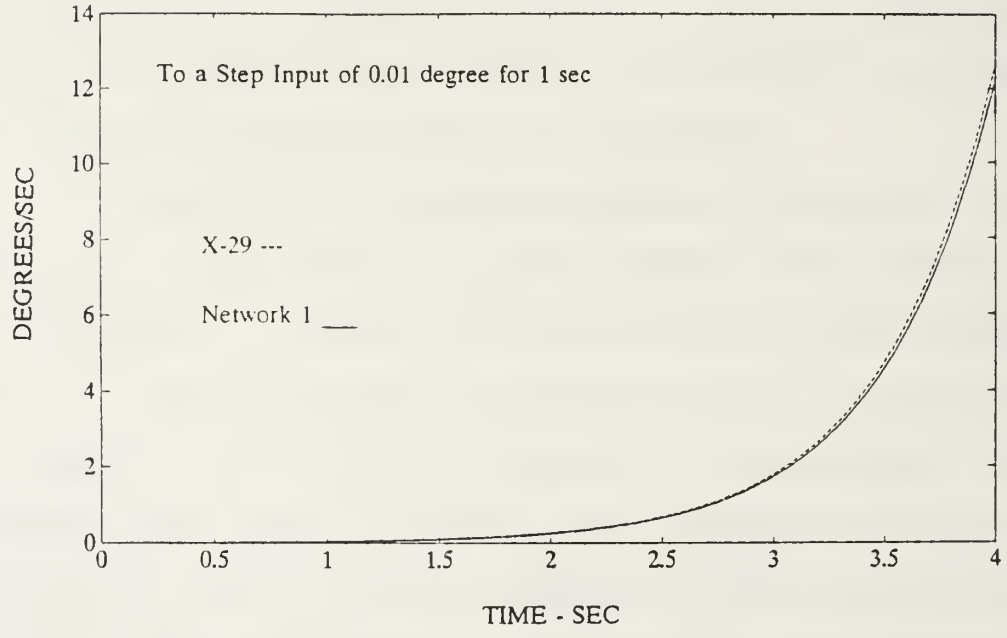


Figure 76 X-29 Plant Model and Network 1 q Time Responses to Input 1

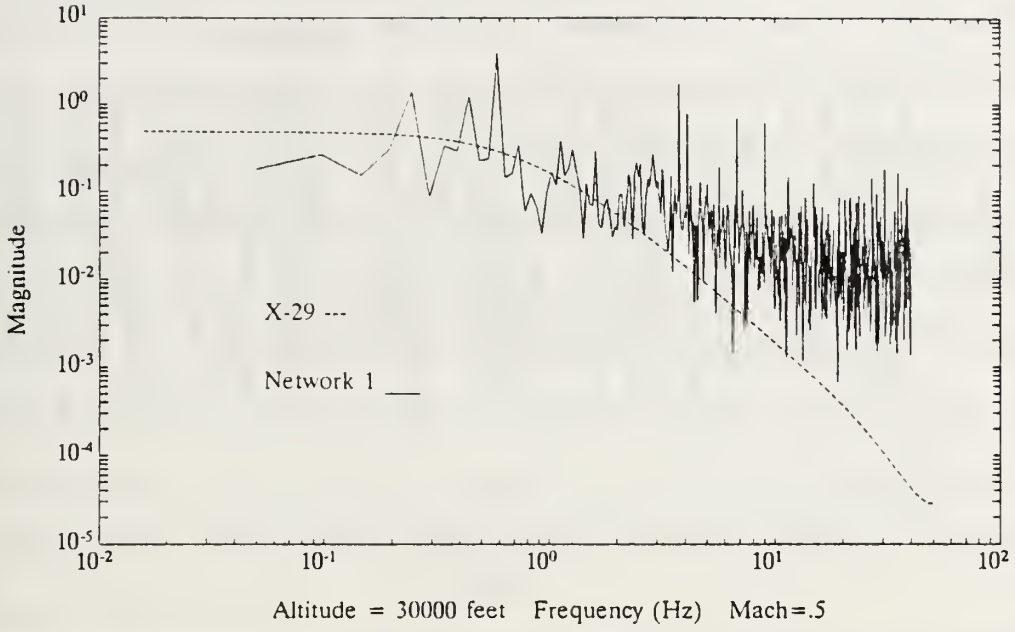


Figure 77 X-29 Plant Model and Network 1 α Frequency Responses to Input 1

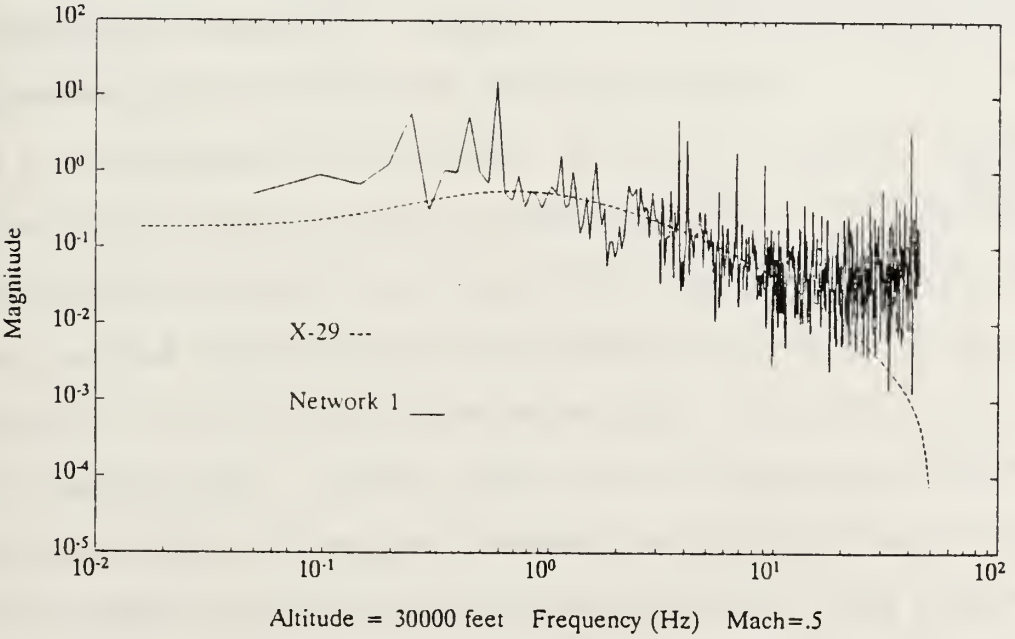


Figure 78 X-29 Plant Model and Network 1 q Frequency Responses to Input 1

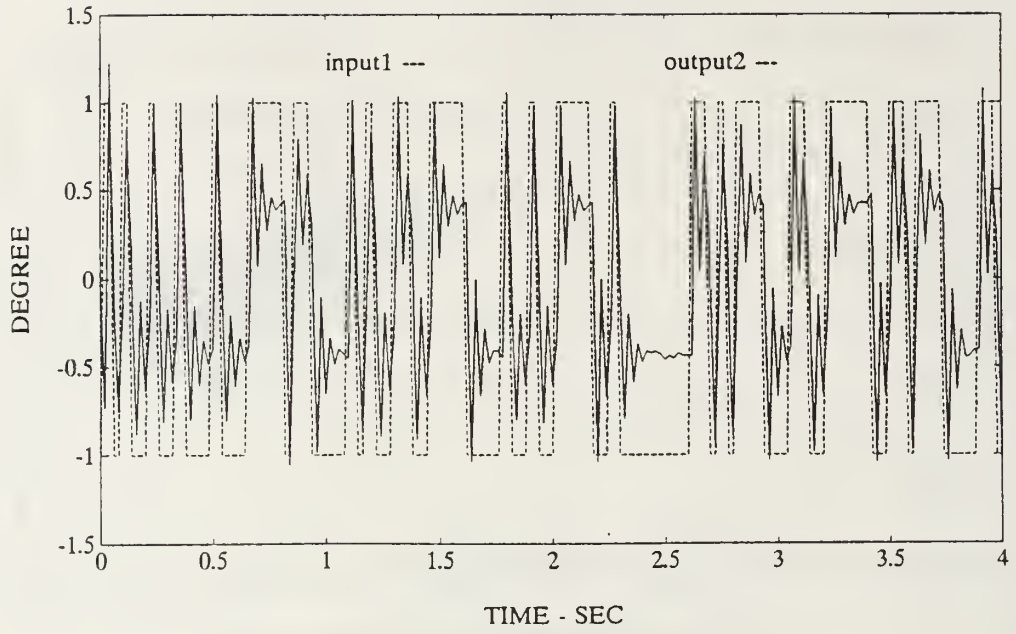


Figure 79 RB Inputs Comparison after 500K Epochs (case #5)

B. CONFIGURATION #3: SIMULATIONS OF THE X-29 EXISTING CONTROLLERS AND THE X-29 PLANT

In the simulation of the optimal X-29 closed-loop plant of case #1 and in the simulation of the limited X-29 closed-loop plant of case #2, the entire closed-loop architecture of Fig. 28 is emulated, whereas in case #6 and #7 each controller and plant is simulated separately as the open-loop architecture of Fig. 33. After both the controller and the plant are emulated, case #8 will close the open-loop model of both cases by connecting in series the neural network representing the controller and the neural network representing the plant, and by feeding back the errors of the plant network outputs to the controller network inputs, as illustrated in Fig. 33. In the

open-loop architecture of Fig. 33, level 1 emulates the X-29 controller and level 2 emulates the X-29 plant. The neural network representation of Fig. 33 is shown in Fig. 34. The first MIMO neural network structure of Fig. 34 simulates the controller and the second MIMO neural network structure of Fig. 34 simulates the plant. The optimal controller case #6 introduces a nonlinear network model, whereas the limited controller case #7 utilizes a linear network model. In this configuration, only the time domain will be analyzed since the interests are on the time responses of both inputs to case #8, as described above.

1. Case # 6 - Simulation of the X-29 Optimal Controller and the X-29 Plant

This case emulates the stable optimal, controller at level 1 and the plant of the X-29 at level 2. This controller is stable since the poles and zeros of the transfer functions of the two inputs are within the unit circle.

The optimal controller is first emulated using a linear neural network i.e. no hidden layers. The linear neural network 1 has not learned to model the limited controller even after 200,000 epochs or 4000 sec, as shown in Fig. 80 through Fig. 83. The network has difficulties in simulating the magnitude of the linear ramps of the responses and the magnitude of the excursions. The best example is shown in Fig. 82. The network could learn the ascent of the first peak, but it could not make the descent on time. The difficulties in

simulating the excursions are due to the fact that their rise times are approaching the sampling time of 0.02 seconds. Nevertheless, in all four graphs the time responses of the controller network, network 1, do not show any time shifts.

Since the optimal controller could not be emulated using a linear network model, nonlinearity was introduced to see if a nonlinear network model could bring better results. Therefore, two non-linear networks were investigated in this case study.

The first nonlinear network involves the addition of one hidden layer between the control input layer and the output layer of the controller MIMO network structure of Fig. 34. The hyperbolic tangent function is used as the transfer function of the hidden layer.

Various numbers of elements in the hidden layer have been tested. The best results were obtained within 25,000 epochs or 500 seconds using 42 elements.

The time responses of the X-29 linear, optimal controller and of the first nonlinear controller network to a step input of one degree for one second are shown in Fig. 84 through Fig. 87. The RMS errors are on the order of 0.0001 for *output 1* or 0.01 percent of the maximum output of one, and 0.00005 for *output 2*. There are some improvements from the linear network responses (0 hidden layer) to the nonlinear network responses (1 hidden layer). Referring to the above mentioned figures, all the linear ramps of the responses are

well modelled. However, the peaks' values are not well represented, as shown in Fig. 86.

The second nonlinear network involves the addition of two hidden layers between the control input layer and the output layer of the controller MIMO network structure of Fig. 34. The first hidden layer from the bottom has the same number of elements as the one of the previous test, i.e., 42. To determine the optimal number of elements in the second hidden layer, a SVD analysis was carried out based on the weight connections of the two hidden layers.

The SVD was calculated for different numbers of elements (30, 21, 12 and 5) or trials using the weight matrices, which are composed of the connections weights of the two layers. The results are given in Fig. 88 through 91. In all four trials, the networks have been trained using the same learning rate. In all SVD plots, three lines stand out, meaning that the optimal number of elements in the second hidden layer should be three. After some oscillations between 0 and 1500 epochs, the network in all four trials (30, 21, 12, and 5 elements) stabilizes to constant values. For example, in Fig. 88 the network comprising a second hidden layer of 30 elements stabilizes around SVD 5, 3.5, and 0.5.

Therefore, forty-two elements in the first hidden layer and three elements in the second hidden layer should be sufficient to represent the second nonlinear network model (2 hidden layers).

The time responses of the X-29 linear, optimal controller and the second nonlinear network 1 to a step input of one degree for one second are shown in Fig. 92 through Fig. 95. The RMS errors are on the order of 0.001 for *output 1* and 0.0004 for *output 2*. The responses are very similar to the first nonlinear network 1 (1 hidden layer). Comparing Fig. 92 with Fig. 84, Fig. 93 with Fig. 85, and Fig. 95 with Fig. 87, the linear ramps of the two nonlinear networks are well modelled but the peaks are not well represented. Comparing Fig. 94 with Fig. 86, both networks responded with high peak values which are not present in the true optimal controller responses. As with the first nonlinear network (1 hidden layer), no time shifts are found.

The second part of case #6 involves the simulation of the X-29 unstable plant at level 2, as shown in Fig. 33. In the X-29 plant of case #5 only one input to the plant was necessary to investigate the case, whereas in case #6, case #7 and case #8 both inputs to the plant are necessary. After 40,000 epochs or 800 seconds, network 2 has learned to model the unstable X-29 plant using a MIMO network structure rather than a SIMO structure as in the X-29 plant case #5. It took as long to train the SIMO network structure as to train the MIMO structure. The q and α time responses of the X-29 plant model and of neural network 2, which emulates the plant, to a step of 0.01 degree for one second applied to input 2 are given in Fig. 96 and 97. The responses to input 1 were given in the X-

29 plant model of case #5. Once again, both responses have an exponential departure with respect to time. This time, the departures are in the opposite direction. As with case #5, the plant outputs were limited to values $\pm$ 1 to control the negative exponential responses.

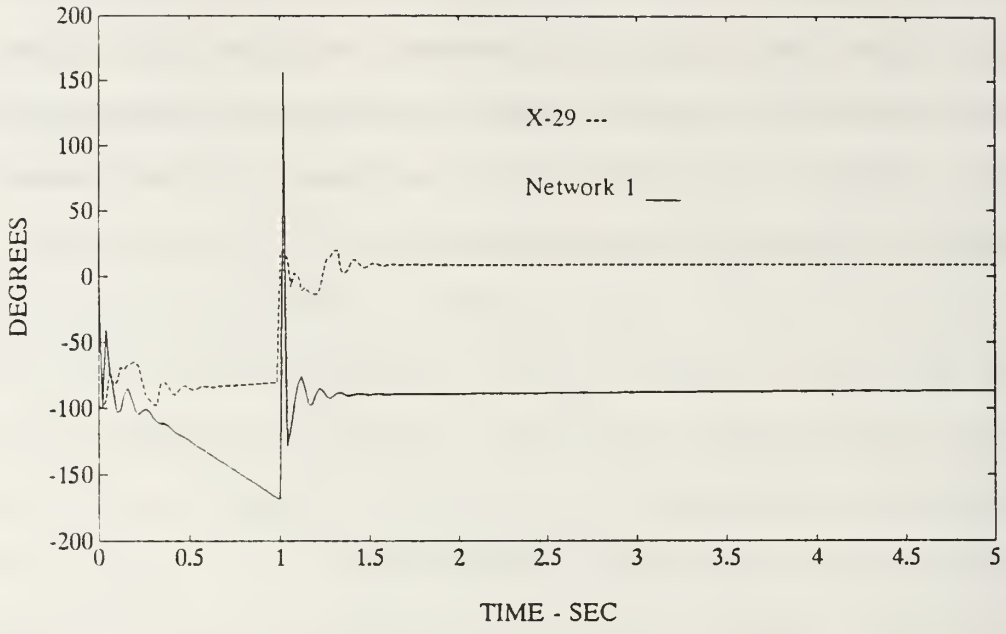


Figure 80 X-29 Optimal Controller Model and Network 1 α Time Responses to Input 1 (0 Hidden layer)

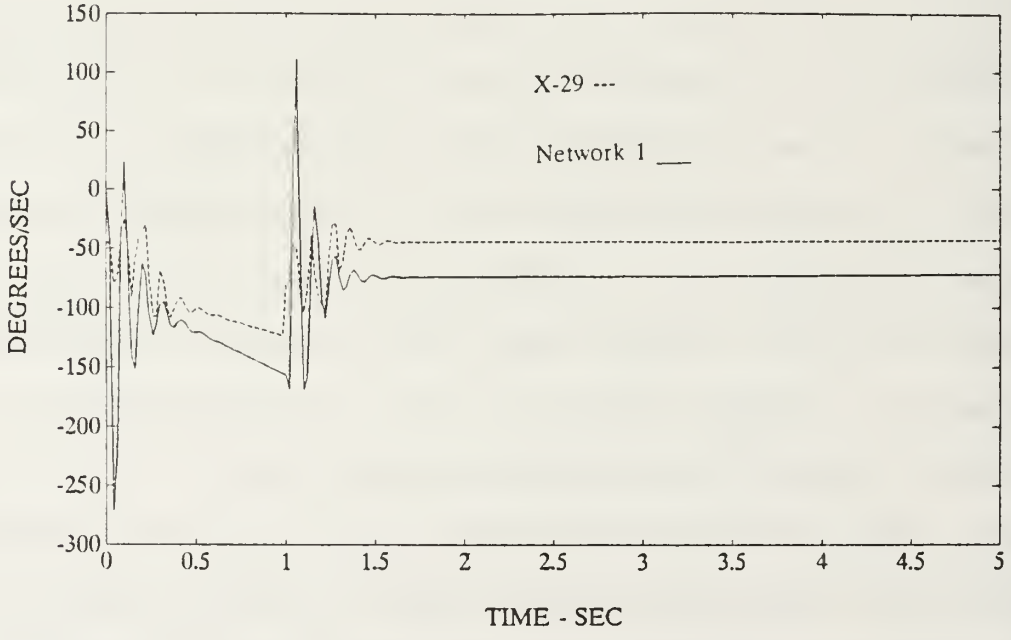


Figure 81 X-29 Optimal Controller Model and Network 1 q Time Responses to Input 1 (0 Hidden layer)

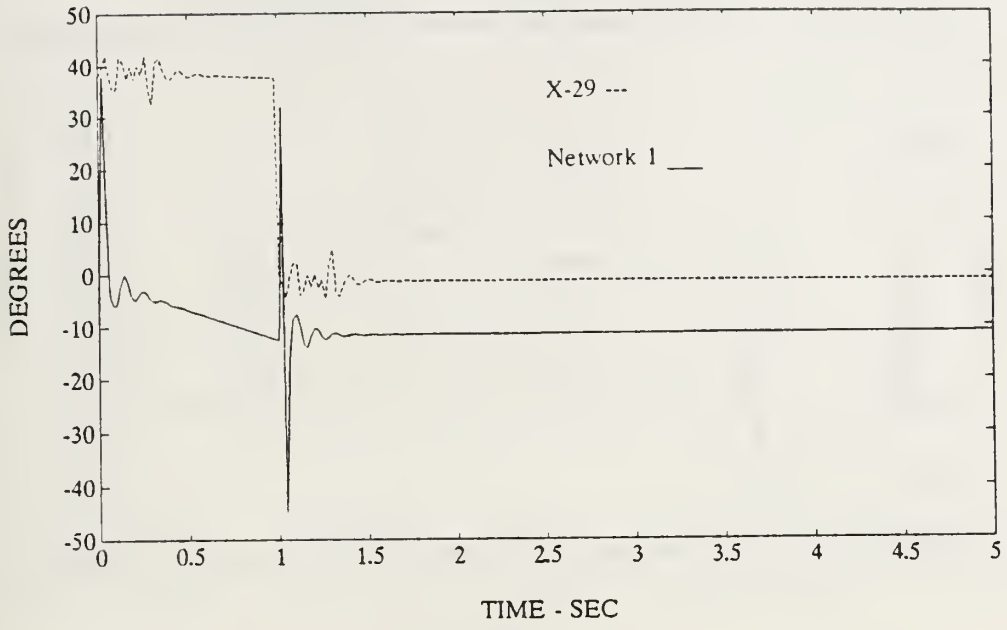


Figure 82 X-29 Optimal Controller Model and Network 1 α Time Responses to Input 2 (0 Hidden layer)

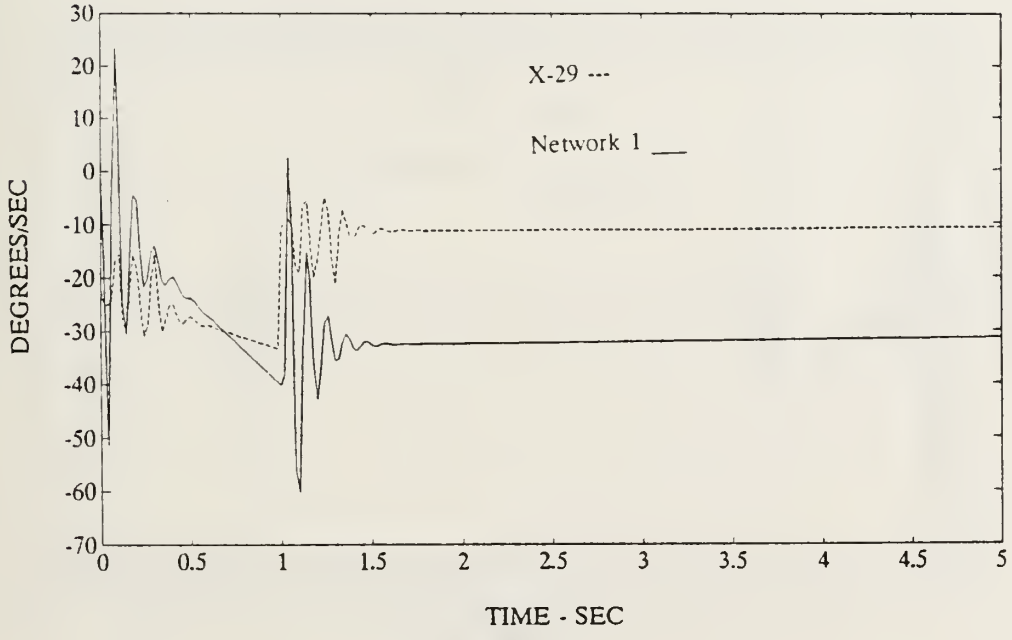


Figure 83 X-29 Optimal Controller Model and Network 1 q Time Responses to Input 2 (0 Hidden layer)

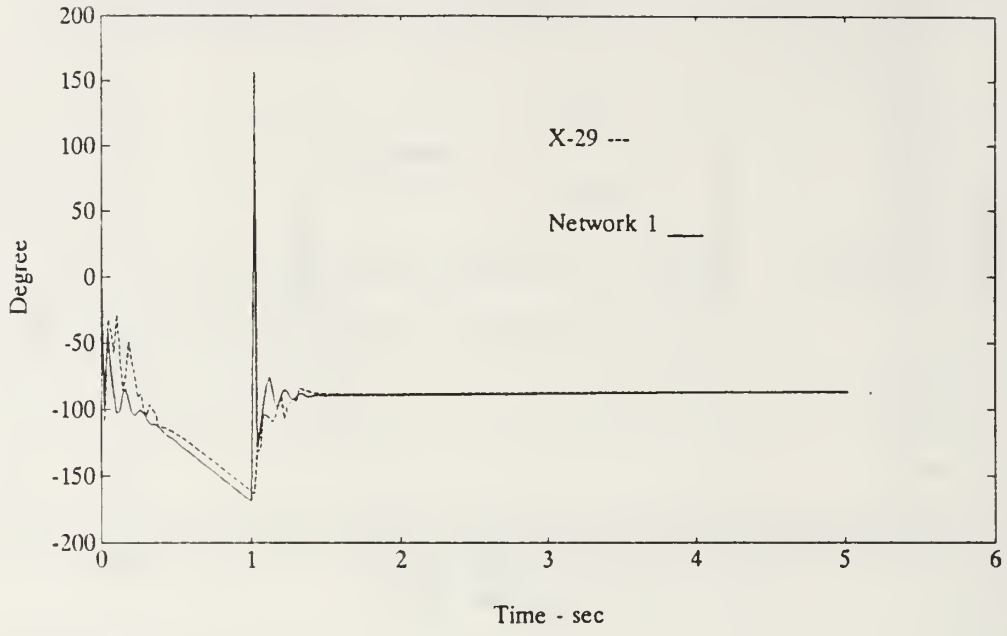


Figure 84 X-29 Optimal Controller Model and Network 1 α Time Responses to Input 1 (1 Hidden layer)

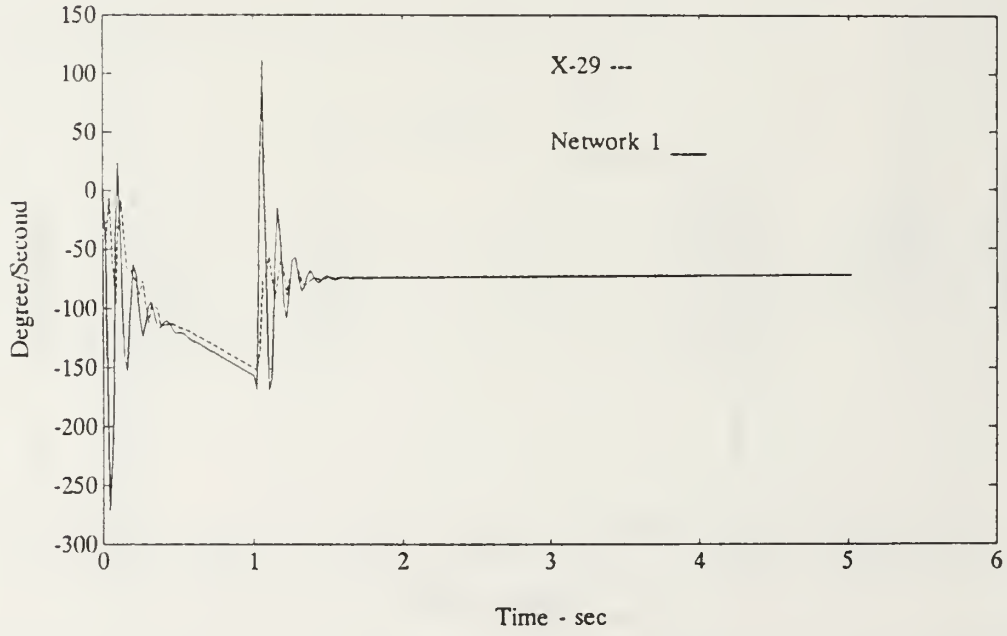


Figure 85 X-29 Optimal Controller Model and Network 1 q Time Responses to Input 1 (1 Hidden layer)

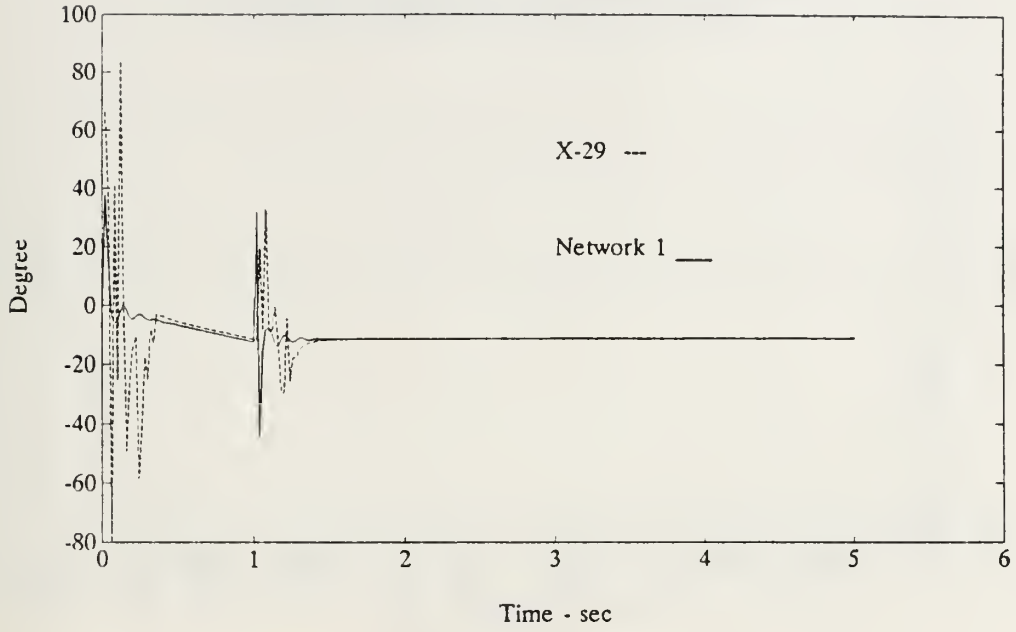


Figure 86 X-29 Optimal Controller Model and Network 1 α Time Responses to Input 2 (1 Hidden layer)

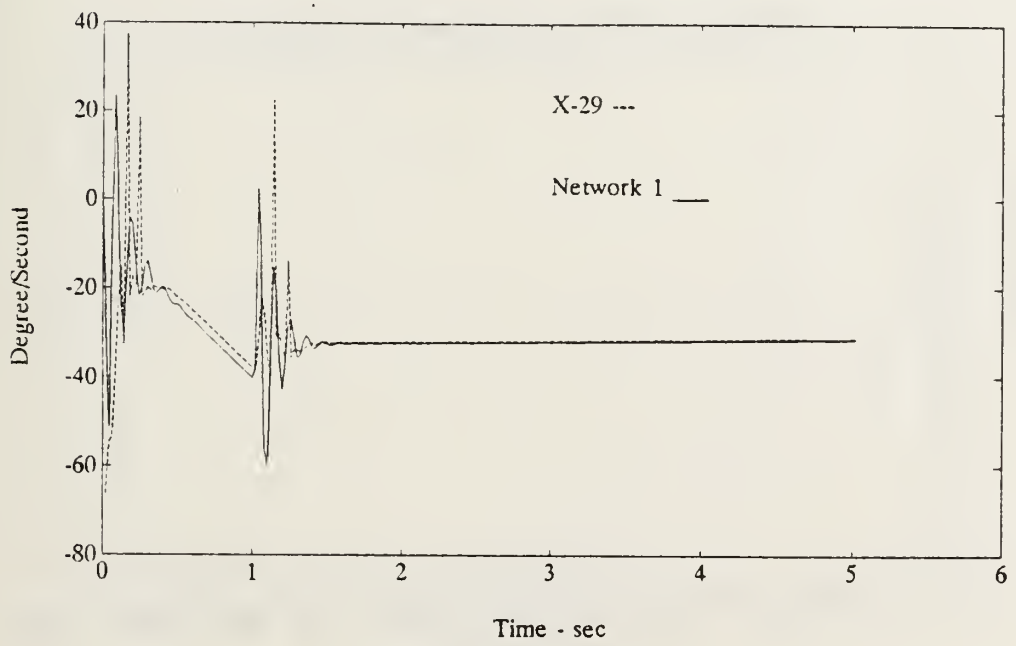


Figure 87 X-29 Optimal Controller Model and Network 1 q Time Responses to Input 2 (1 Hidden layer)

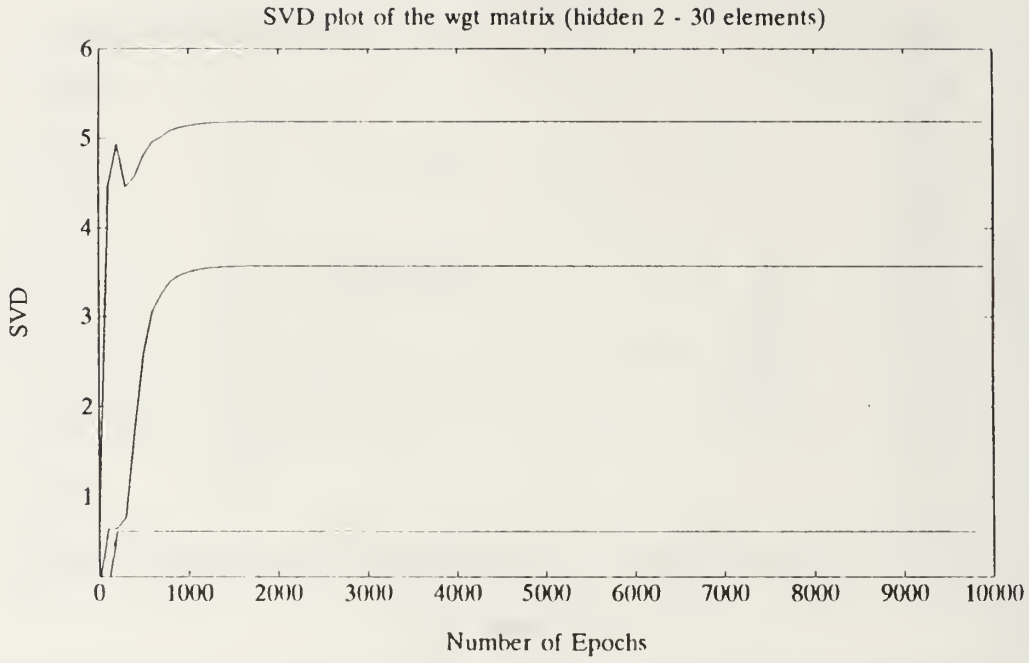


Figure 88 SVD Plot of the Wgt Matrix (Hidden 2- 30 elements)

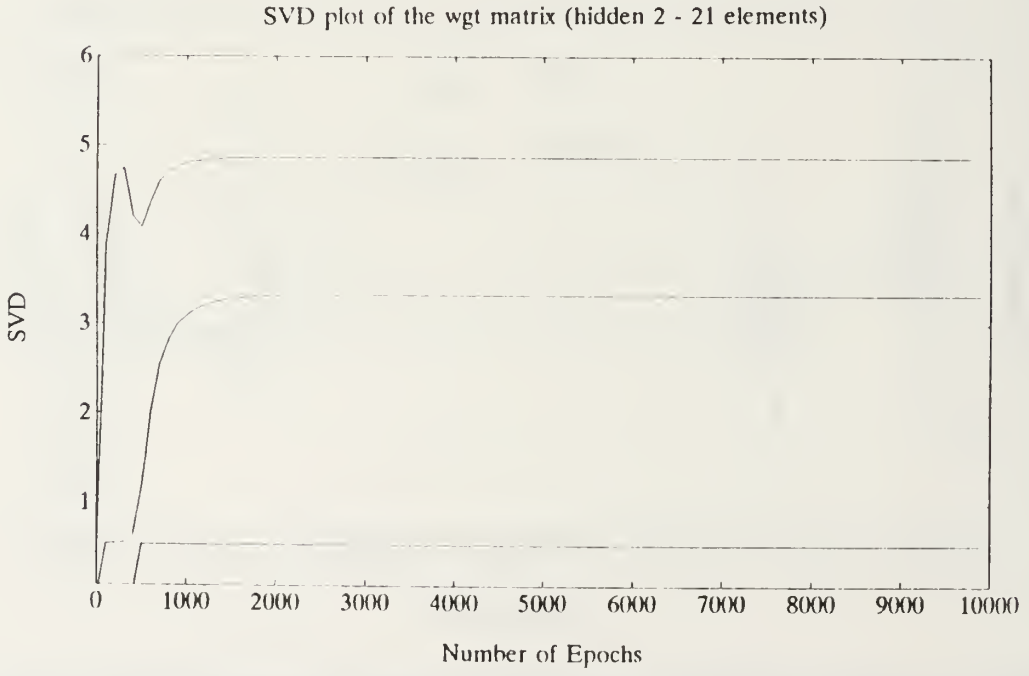


Figure 89 SVD Plot of the Wgt Matrix (Hidden 2- 21 elements)

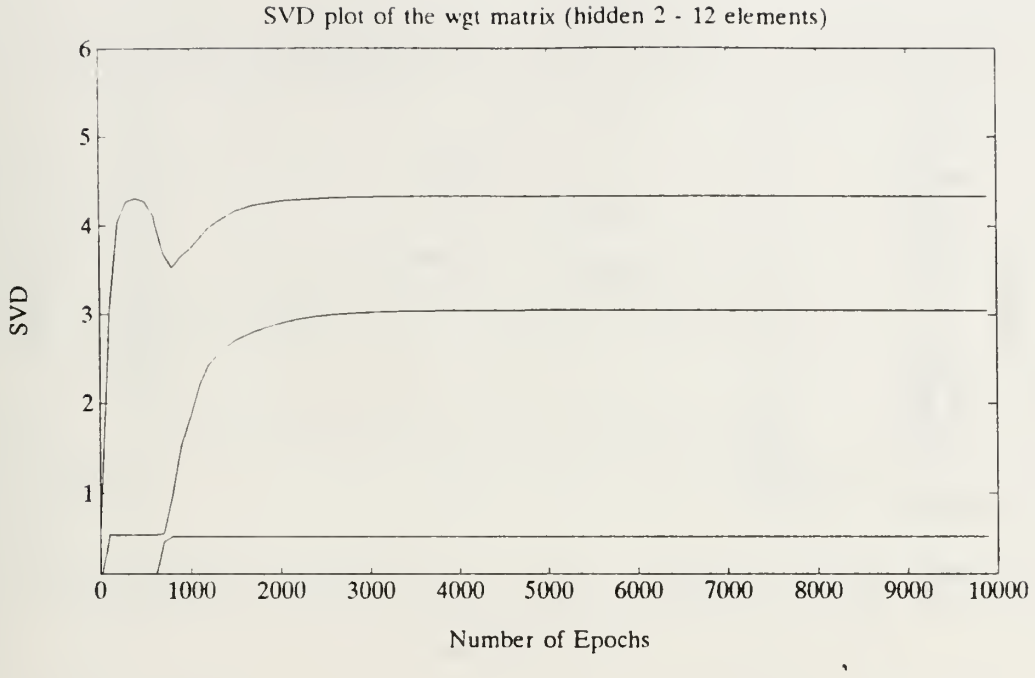


Figure 90 SVD Plot of the Wgt Matrix (Hidden 2- 12 elements)

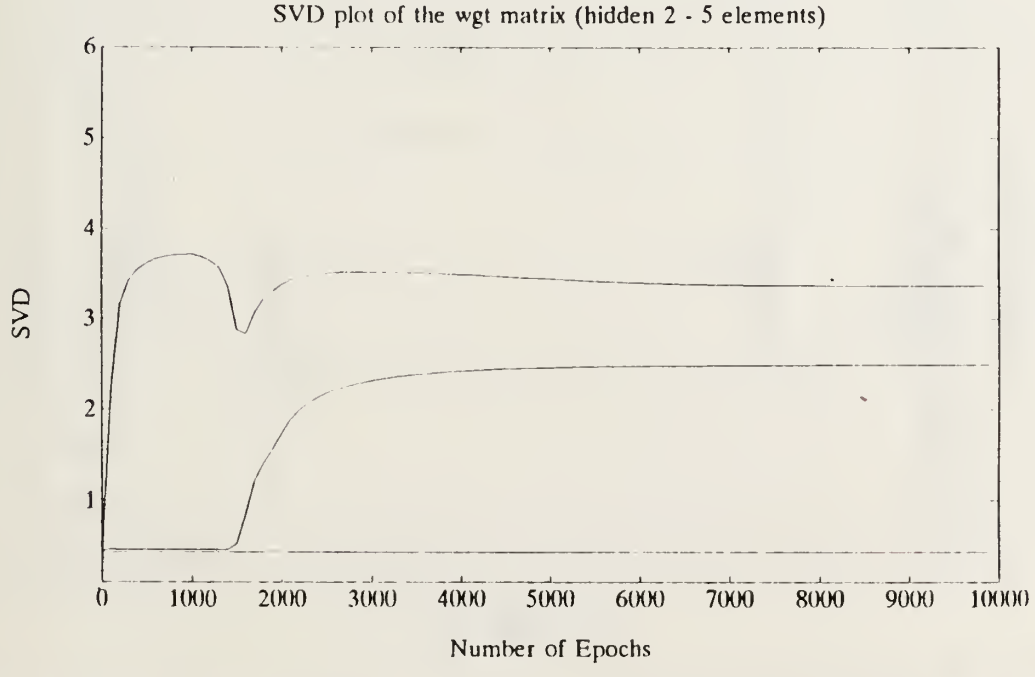


Figure 91 SVD Plot of the Wgt Matrix (Hidden 2- 5 elements)

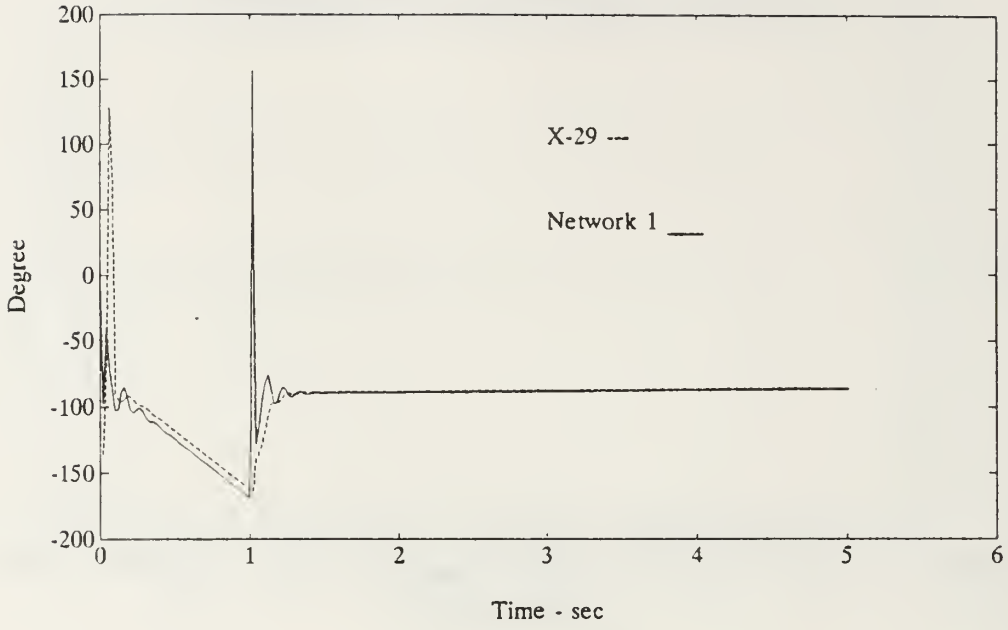


Figure 92 X-29 Optimal Controller Model and Network 1 α Time Responses to Input 1 (2 Hidden layer)

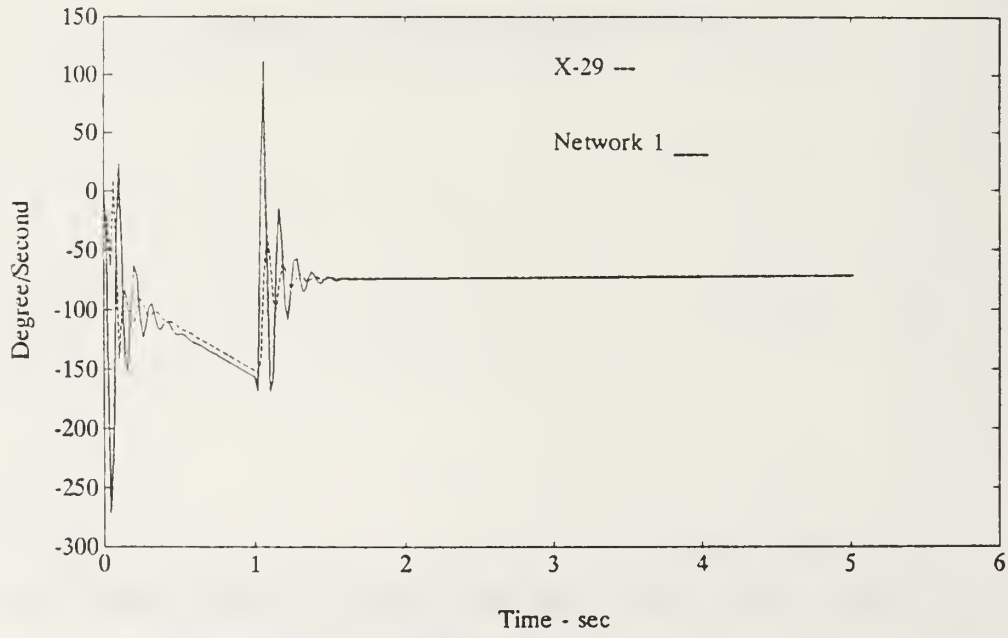


Figure 93 X-29 Optimal Controller Model and Network 1 q Time Responses to Input 1 (2 Hidden layer)

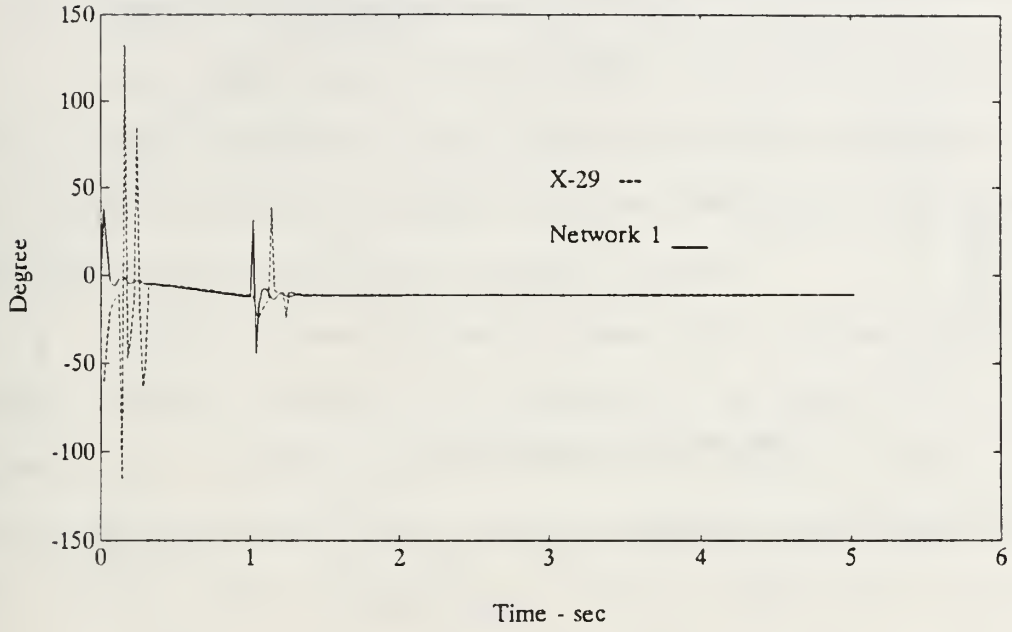


Figure 94 X-29 Optimal Controller Model and Network 1 α Time Responses to Input 2 (2 Hidden layer)

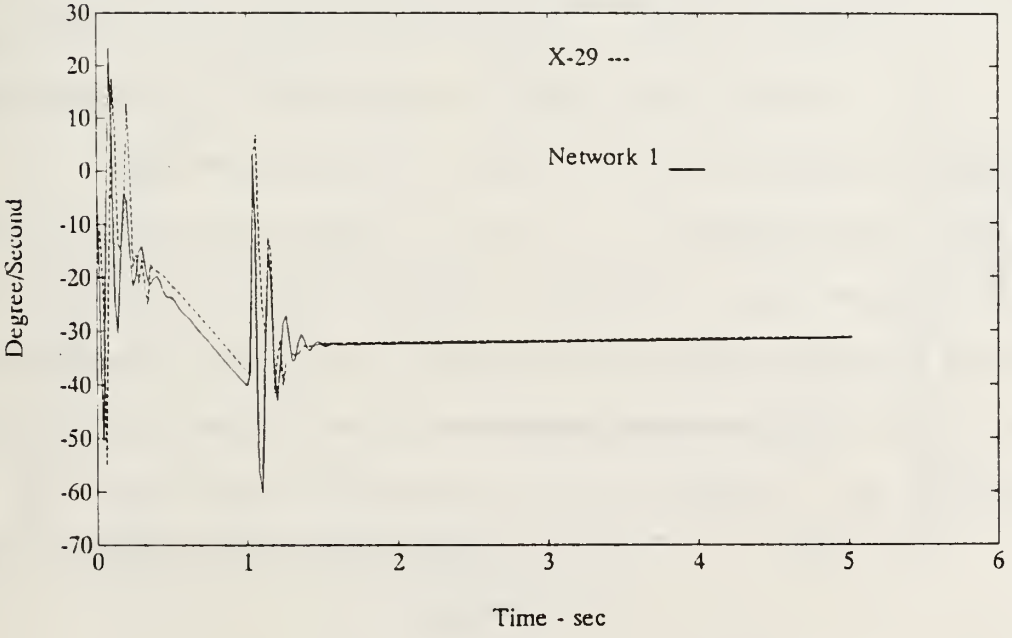


Figure 95 X-29 Optimal Controller Model and Network 1 q Time Responses to Input 2 (2 Hidden layer)

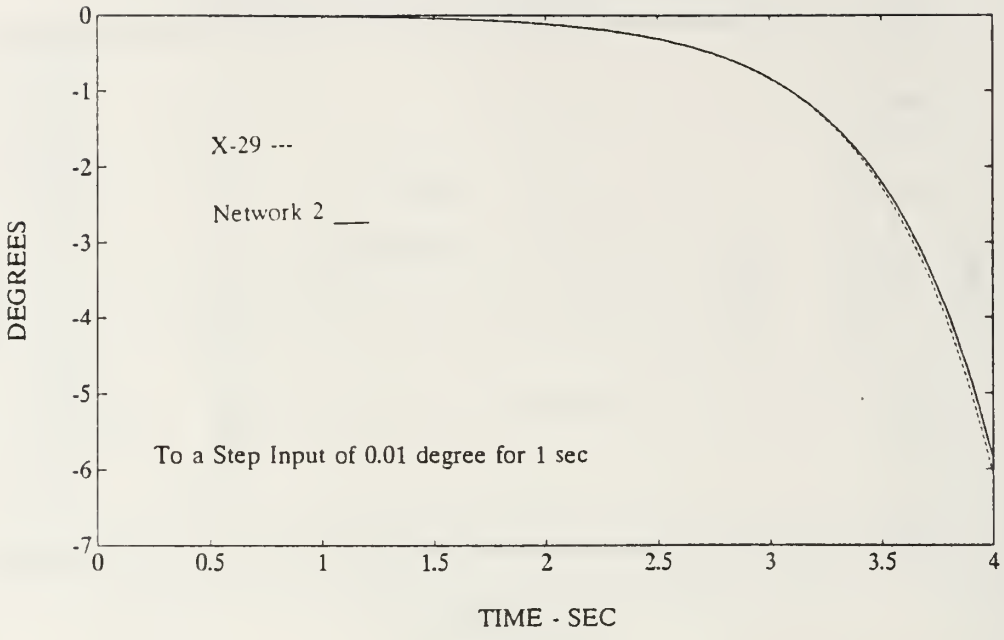


Figure 96 X-29 Plant Model and Network 2 α Time Responses to Input 2

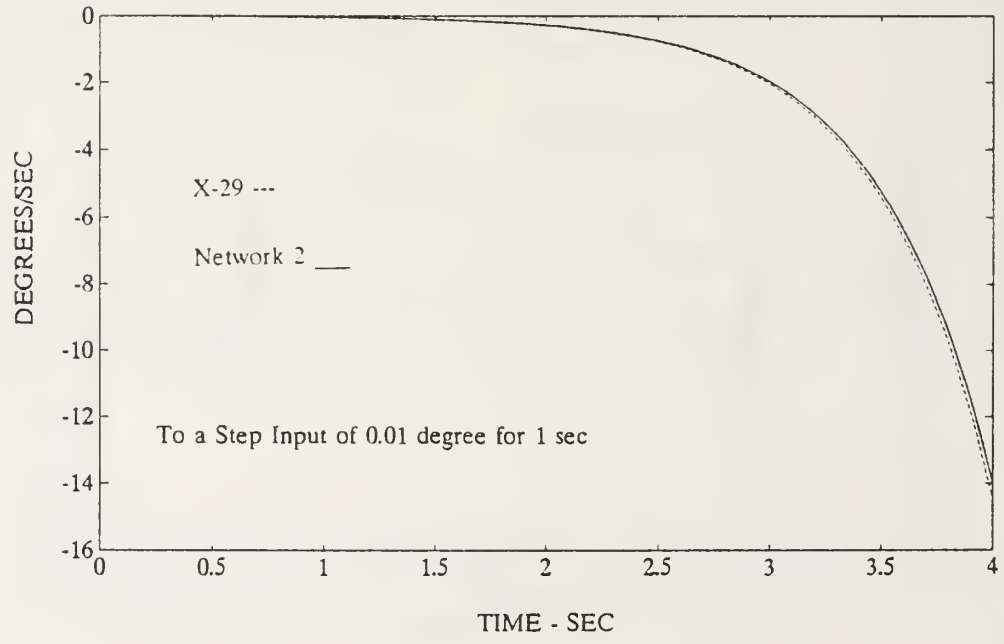


Figure 97 X-29 Plant Model and Network 2 q Time Responses to Input 2

2. Case # 7 - Simulation of the X-29 Limited Controller and the X-29 Plant

This case emulates the stable, limited controller at level 1 and the plant of the X-29 at level 2, as shown in Fig. 33. The controller is stable because the poles and zeros of the two inputs are within the unit circle.

Network 1 has learned to model the limited controller within 20,000 epochs or 400 seconds, as shown in Fig. 98 through 101. The RMS errors are on the order of 0.001 for α and on the order of 0.0005 for q. Comparing the four figures, the α and q time responses to input 1 show near to exact solutions, whereas the α and q time responses to input 2 show minor deviations from the true limited controller model. Further training did not better the results of input 2. Nonlinearity was also introduced to network 1 emulating the limited controller by adding hidden layers , but no improvement to the present results were found.

The second part of case #7 involves the simulation of the X-29 unstable plant at level 2, as shown in Fig. 33. After 40,000 epochs or 800 seconds, network 2 has learned to model the unstable X-29 plant using the same MIMO network structure than the optimal case #6. As with case #6, the output values of the plant were limited to ± 1 to control the exponential departures of the responses.

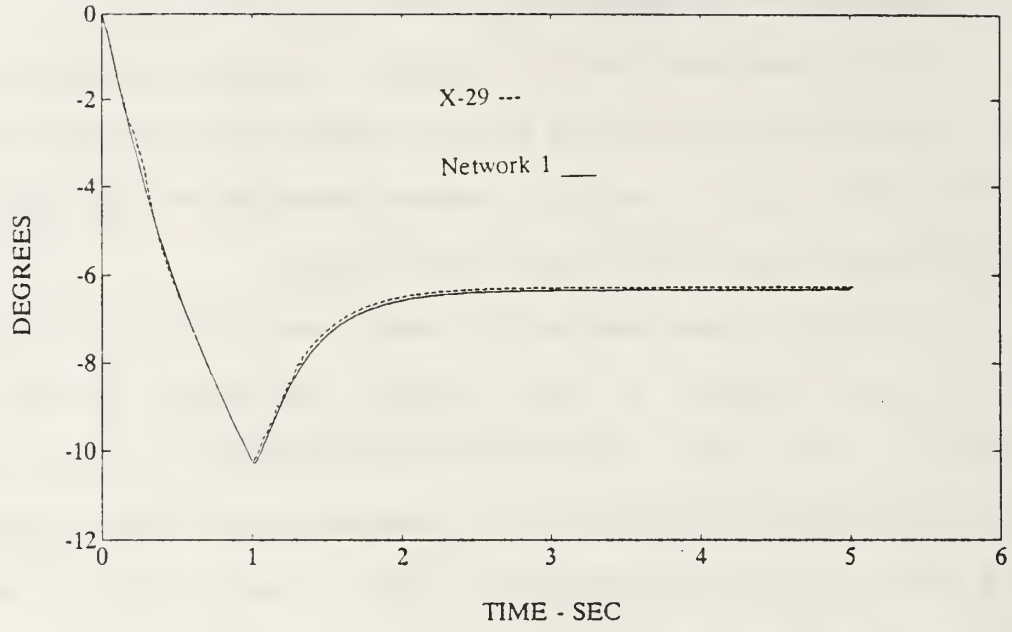


Figure 98 X-29 Limited Controller Model and Network 1 α Time Responses to Input 1

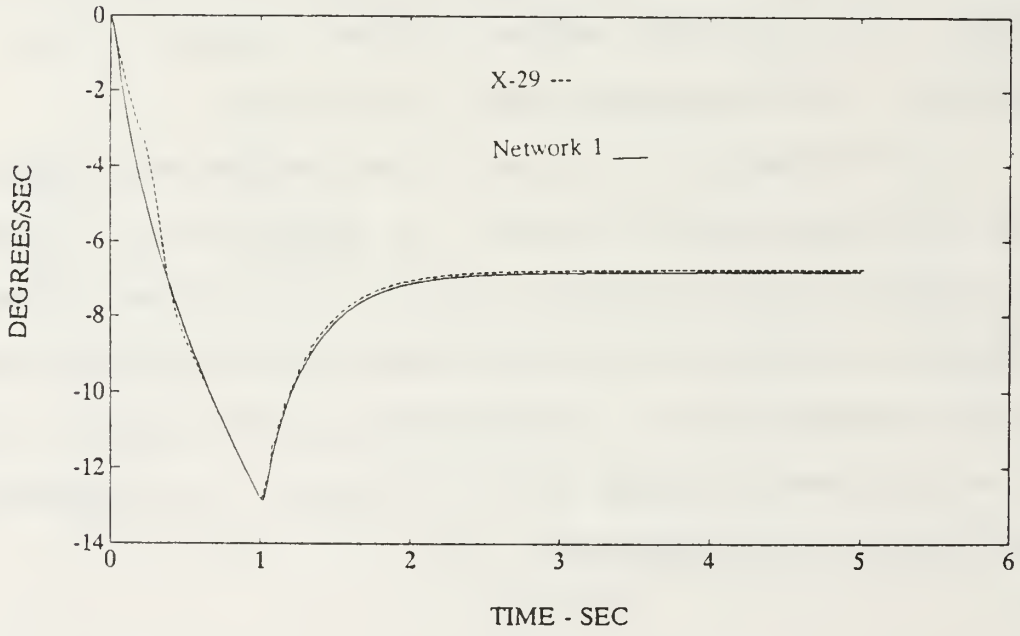


Figure 99 X-29 Limited Controller Model and Network 1 q Time Responses to Input 1

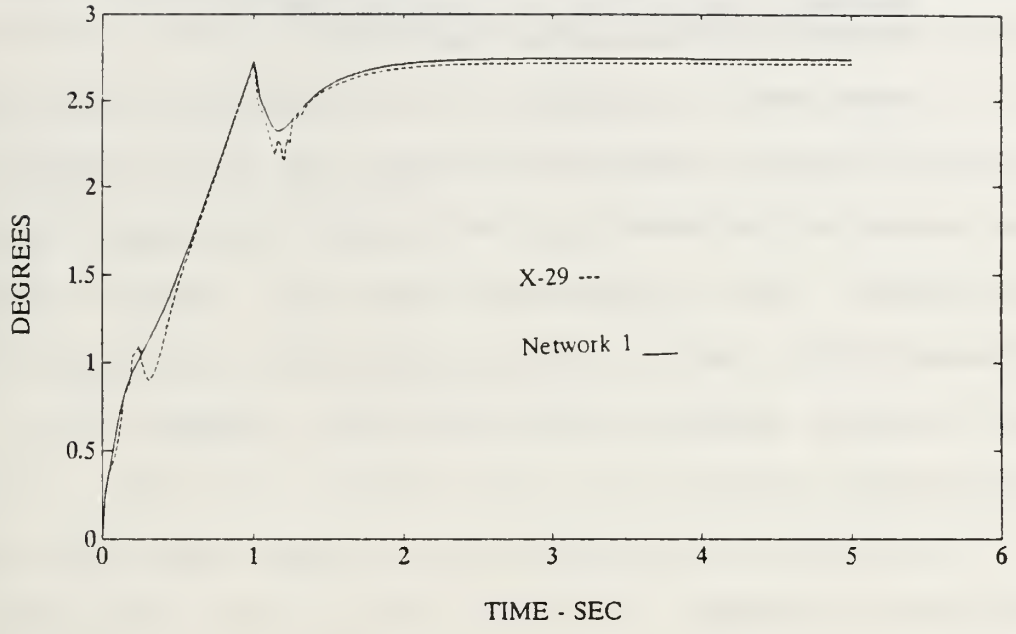


Figure 100 X-29 Limited Controller Model and Network 1 α Time Responses to Input 2

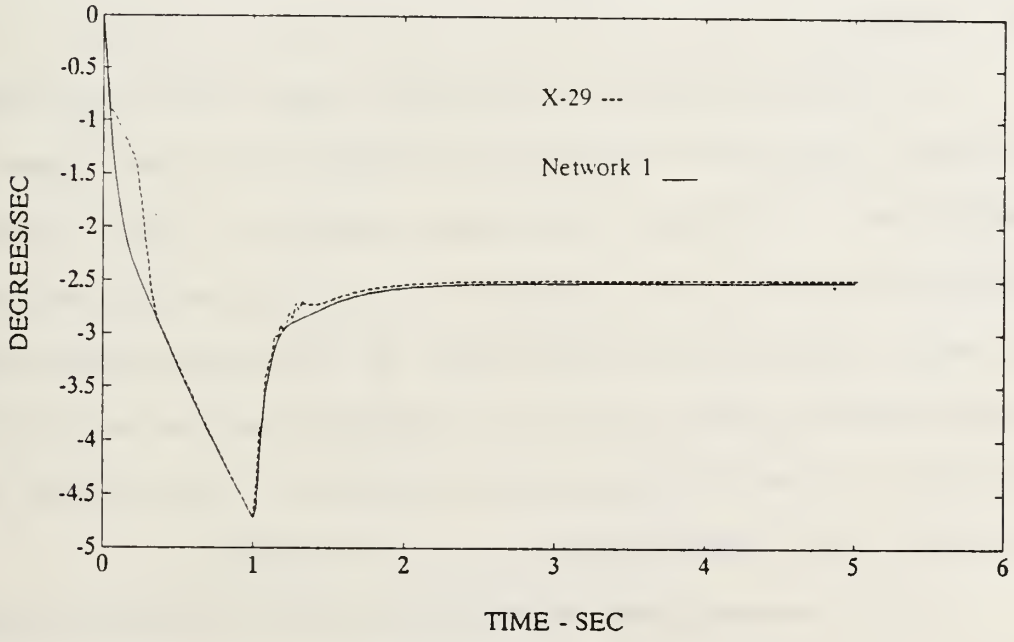


Figure 101 X-29 Limited Controller Model and Network 1 q Time Responses to Input 2

3. Case #8 - Closure of the Open-Loop Model of the Optimal Controller of Case #6 and of the Limited Controller of Case #7

After both the controller and the plant of case #6 and #7 are emulated, case #8 will close the open-loop model of both cases by connecting in series the neural network representing the controller and the neural network representing the plant, and by feeding back the errors of the plant network outputs to the controller network inputs, as illustrated in Fig. 33.

Case #8 is divided in two parts. The first part examines the closure of the open-loop model of the optimal controller of case #6, and the second part examines the closure of the open-loop model of the limited controller of case #7. These operations are performed in case #8 of the USERIO program.

In the first part of case #8, since the X-29 optimal controller could not be modelled exactly by the linear (0 hidden layer) or the two nonlinear networks (1 and 2 hidden layers) of case #6, no solution to the closure of the open-loop model, case #8, was obtained. As mentioned previously, the peak values could not be well represented in the optimal controller of case #6 since the excursions' rise times were too close to the sampling time of 0.02 seconds.

In the second part of case #8, since the linear neural network 1 of case #7 has learned to model the limited controller very well, a solution to the closure of the open-

loop model was obtained. The limited performance X-29 closed-loop model and the networks' α and q time responses to input 1 and 2 are given in Fig. 102 through 105. Referring to Fig. 98 through 101 of the limited performance case #7, the minor deviations of the network time responses from the true model explain the small time shifts occurring in the closure of the open-loop model responses, as shown in Fig. 102 through 105.

In summary, the third configuration could simulate case #7, the X-29 limited controller and the X-29 unstable plant, and case #8 part II, the closure of the open-loop model of case #7, with high accuracy. However, the third configuration could not simulate the optimal controller of case #6 sufficiently to permit case #8 part I, the closure of the open-loop model of case #6, to take place.

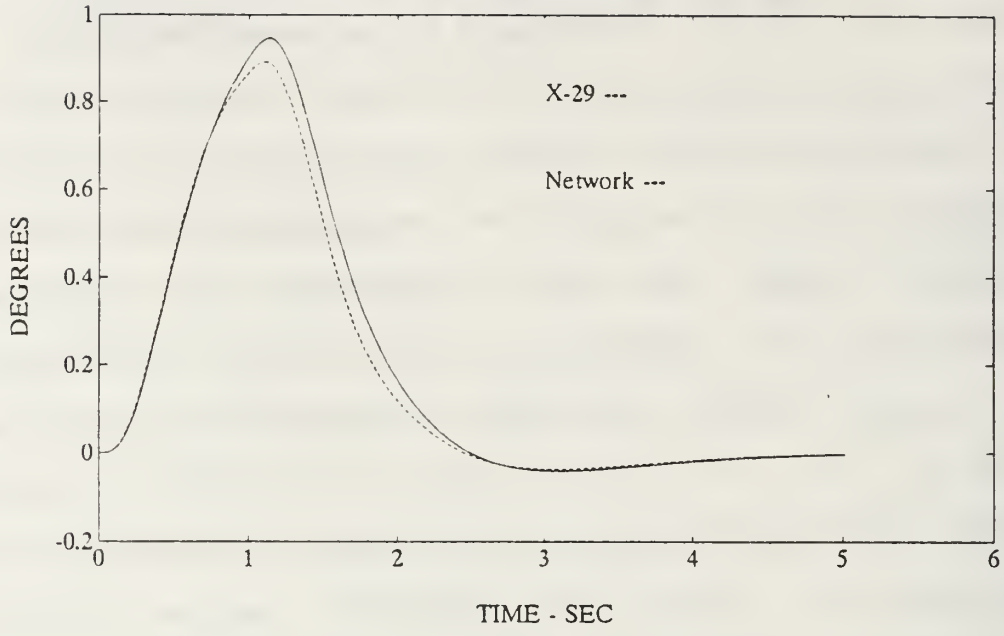


Figure 102 X-29 Closed-Loop Model and Networks α Time Responses to Input 1 (Limited case)

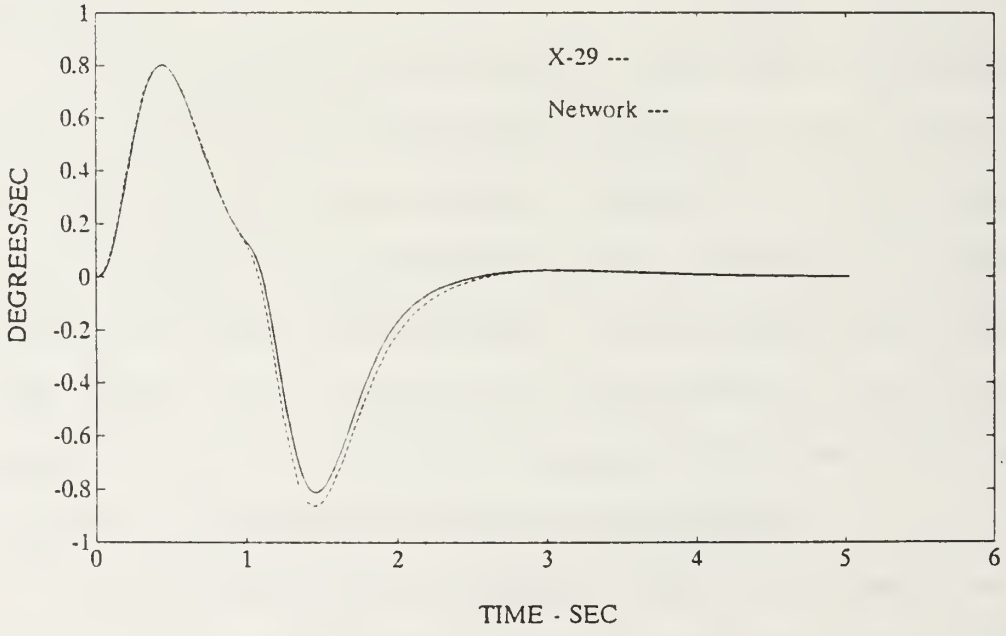


Figure 103 X-29 Closed-Loop Model and Networks q Time Responses to Input 1 (Limited case)

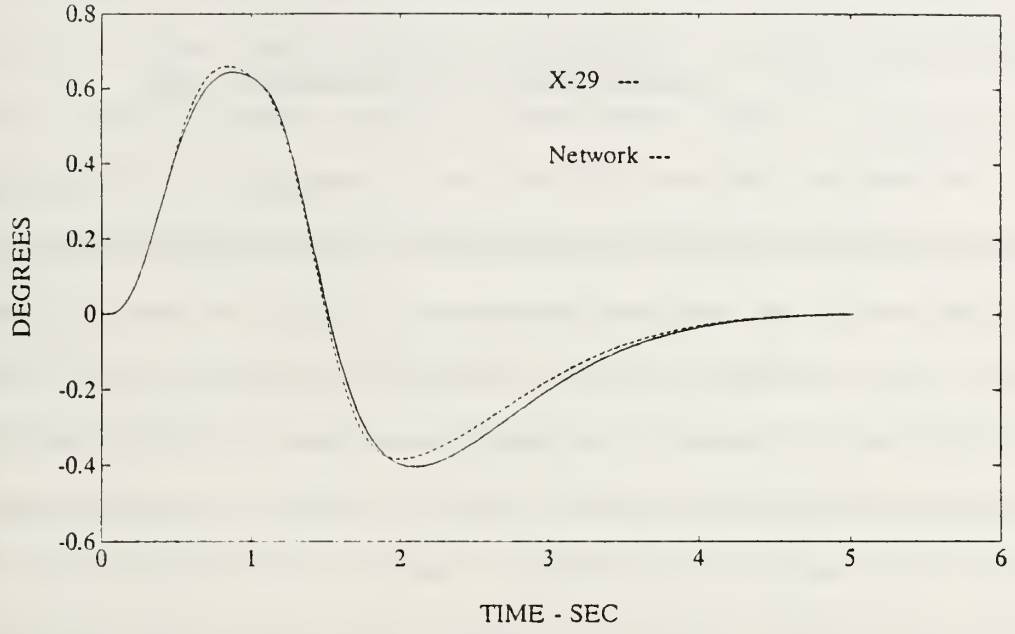


Figure 104 X-29 Closed-Loop Model and Networks α Time Responses to Input 2 (Limited case)

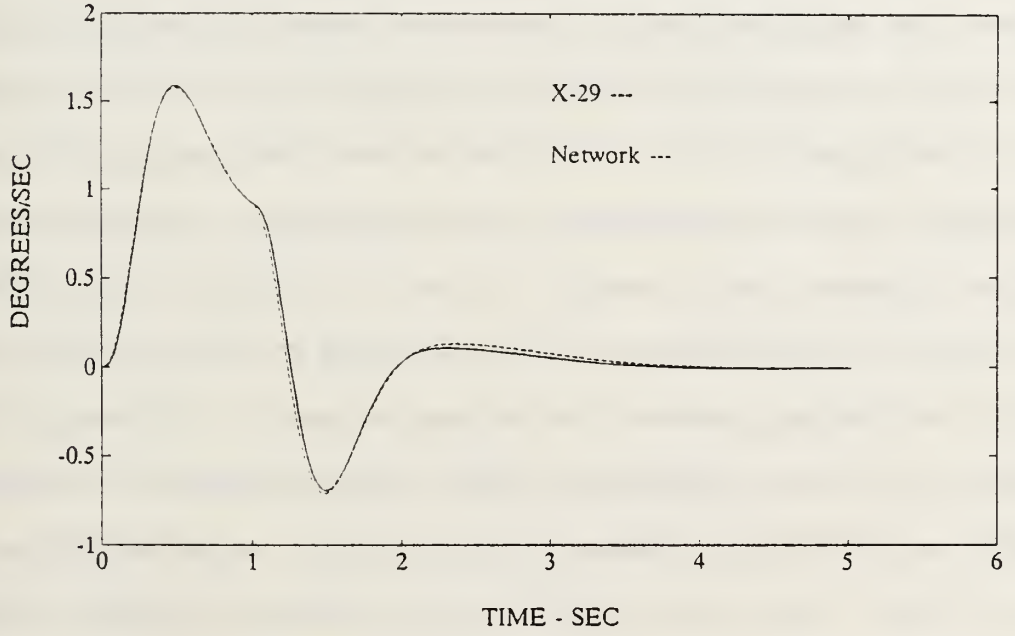


Figure 105 X-29 Closed-Loop Model and Networks q Time Responses to Input 2 (Limited case)

VII. CONCLUSIONS AND RECOMMENDATIONS

The neural network structures developed in this thesis demonstrate the ability of parallel distributed processing in solving adaptive control problems. Adaptive control theory implied a combination of a control method and a model estimation. The control method chosen was the Lyapunov Model Reference Adaptive Control (MRAC) in which the system was forced to follow the reference model with zero error. The controller itself, the weighted one step ahead prediction controller, involved the weighted sum of the state variables and the reference input. The model estimation chosen was the linear least square estimate in which the predicted output became the weighted sum of the terms in the regression vector. These weights were adjusted by minimizing the error between the network and the true X-29 responses. The implementation of the neural network adaptive control structure was demonstrated on the longitudinal dynamics of the X-29 fighter aircraft.

Three configurations were proposed to train the neural network adaptive control structures to provide the appropriate inputs to the unstable X-29 plant so that desired responses could be obtained. These configurations were presented in eight cases. The first configuration representing the closed-loop architecture of Fig. 28 could simulate, with a linear network model, the large order and stable optimal and limited

performance X-29 closed-loop plants with high accuracy. The networks' time and frequency responses of both performance cases, case #1 and case #2, developed near to exact model solutions.

The second configuration representing the inverse plant architecture of Fig. 29 could simulate in both time and frequency domain the large order and stable optimal performance X-29 inverse closed-loop transfer function, case #3, and the small order, unstable A-4D inverse plant, case #4. However, the second configuration could not simulate, neither with a linear nor a nonlinear network model, the more unstable X-29 inverse plant, case #5.

Since the degree of instability of the X-29 inverse plant is much higher than the degree of instability of the A-4D inverse plant, the simulation of the inverse plant of the A-4D aircraft could be achieved easily compared to the X-29 aircraft, whose inverse plant could not be simulated.

The last configuration representing the open-loop architecture of Fig. 33 could, with a linear network model, simulate the X-29 limited controller and the X-29 unstable plant, case #7, with high precision. In the optimal controller of case #6, the nonlinear neural networks, which were used to model the linear system, performed better than the linear network model.

The use of the SVD analysis was successful in determining the optimal number of elements in the hidden layers.

Further studies are needed to develop improved combinations of linear and nonlinear neural network structures. In addition, it is important to pursue research on ways of reducing the computational time by means of selecting the proper number of elements in the hidden layers or by means of selecting the proper learning rates.

REFERENCES

1. Scott, R. W., *Application of Neural Networks to Adaptive Control*, Aeronautical Engineer Thesis, Naval Postgraduate School, Monterey Ca., December 1989.

2. Rogers, W. L., *Application of Modern Control Theory Synthesis to a Super-Augmented Aircraft*, Aeronautical Master Thesis, Naval Postgraduate School, Monterey Ca., June 1989.

3. Wasserman, P. D., *Neural Computing: Theory and Practice*, Anza Research, Inc., 1989.

4. Klimasauskas, C., and others, *Neuralworks Professional II Manual*, Neuralware, Inc., 1988.

5. Kraft, L. G., and Campagna, D. P., "A Comparison Between CMAC Neural Network Control and Two Traditional Adaptive Control Systems", *IEEE Control Systems Magazine, April 1990*, The Institute of Electrical and Electronic Engineers, Inc., 1990.

6. Goodwin, G. C., and K. S. Sin, *Adaptive Filtering: Prediction and Control*, Prentice-Hall, Inc., 1984.

7. Ljung, L., *System Identification: Theory for the User*, Prentice-Hall, Inc., 1987.

8. *Jane's All the World's Aircraft 1987-1988*, pp. 430-432, Jane's Information Group Limited, 1988.

9. *Aviation Week & Space Technology, January 7, 1985*, pp. 47-48, McGraw Hill Publication, 1985.

10. W.L. Rogers and D.J. Collins,"*X-29 H_∞ Controller Synthesis*",paper presented ...

11. Chiang, R. Y., and Safonov, M. G., *Robust-Control Toolbox, User's Guide*, pp. R4-R33, The Mathworks, Inc., June 1988.

12. Sun Microsystems, Inc., *The SPARC Station II* ,1991.

13. Moler, C., J. Little, and S. Bayert, *Pro-MATLAB User's Manual*, The Mathworks, Inc., 1987.

14. Miller, W. T., and others, *Neural Networks for Control*, The MIT Press, 1990.

APPENDIX A: NEURALWORKS PROFESSIONAL II USERIO PROGRAM

```
/*****************************************************************************
 * Source:       simoMonika.c
 * Executable:   simoMonika
 * Version:      3.1
 * Date:         30 August 1991
 * Author:       D. Bertrand
 * Project:      Neural Networks in Adaptive Control
 * Environment:  UNIX/SunOS C
 * Path:         eileen:/fil669/home/Monika
 * Description:  This is a prototype for the USERIO program spawned by
 *               NWORKS Professional II to provide input and output
 *               vectors for the use of an adaptive control neural network.
 *               The program operates by running a simulation of the
 *               longitudinal dynamics of the X-29 and A-4 aircrafts at the
 *               same speed as sampling time of the network.
 *               The program can be used for any SIMO and MIMO linear and
 *               non-linear models.
 *               This USERIO program has been divided into cases:
 *
 *                < Level 1 & 2 represent two networks superimposed >
 *
 * case (c#)     level 1            level 2     control strat.      remarks
 *  ------   ---------------       -------     ------------   ----------------
 *    1,2     X-29, 30 states         ---         contstr1       only Level 1
 *           closed-loop conf.                                    is trained
 *          (optimal & limited)
 *
 *     3      X-29, 30 states      X-29 inverse   contstr1    to train Level 1
 *           closed-loop conf.   closed-loop conf.
 *              (optimal)                        contstr2    to train Level 2
 *
 *     4       A4 plant           Inverse Plant   contstr1   to train Level 1
 *                                                contstr2   to train Level 2
 *
 *     5       X-29 Plant         Inverse Plant   contstr1   to train Level 1
 *                                                contstr2   to train Level 2
 *
 *    6,7     X-29 Controllers     X-29 Plant     contstr2   to train each Level
 *           (Optimal & Limited)                                 individually
 *
 *     8      " " " " " " " "      " " " " "       " " " "   to test case 6,7:
 *                                                connect in series & feedback
 *
 *
 *  case 3,4,5
 *  ----------  In these three cases, two major operations are performed :
 *                -connecting the plant and the inverse plant networks in series
 *                -presenting the desired output to the inverse plant network.
 *              These operations are accomplished in the case request
 *              RQ_LEARNRSLT when identifying the output layer,
 *
 *               if(IOCOUNT == NUM_OUT && IOLAYER == out_lay){
 *                ... for (i=0; i<num_out; i++){
 *                      command2[i]=out[i]; ..meaning that alpha(t) and q(t)
 *                                           become the input of the inverse
 *                                           plant.
 *                   }
 *                   out2[0]=command[0];  ..meaning that r(t-1), the ref.input
 *                                         becomes the desired output of the
 *                                         inverse plant.
```

```
*               }...
*
*  case 6 & 7
*  ---------- In these two cases, both networks are trained separately.
*             The connections between the networks will be done in case #8.
*
*  case 8
*  ------     In this case, two major operations are performed on case #6 & 7:
*               -connecting the controller and the plant networks in series,
*               -feeding back to the controller inputs the error between the
*                network plant outputs and the reference inputs
*             These operations are accomplished in the case request
*             RQ_LEARNRSLT when identifying the plant and the controller layers
*
*              if(IOCOUNT == num_out && IOLAYER==out_lay){
*               ... for(i=0; i<num_out; i++){
*                     command2[i]=out[i]; ..meaning that the controller outputs
*                                           become the plant inputs.
*                   }
*              }...
*
*              if(IOCOUNT == num_out2 &&  IOLAYER==out2_lay){
*              ...for(i-0; i<num_out2; i++){
*                   command[i]=command[i]-out2[i];..meaning that the controller
*                                                   inputs become the error
*                                                   between the plant outputs
*                                                   and the reference in puts.
*                 }...
*              }...
*
*
*             Three different inputs are available:
*
*                 Input 1 - Random Binary
*                 Input 2 - test input 1 with a pulse of 1 degree for 1 sec
*                 Input 3 - test input 2 with a pulse of 1 degree for 1 sec
*
*********************************************************************************

/* Include the following external modules */

#include <stdio.h>
#include <math.h>
#include "userutl.h"
/* #include "transfer30hp.h"          File of parameters for case 1 (optimal) */
/* #include "transfer30lim.h"         File of parameters for case 2 (limited) */
/* #include "transferhp1inv.h"        File of parameters for case 3 (X29)*/
/* #include "transferA4inv.h"         File of parameters for case 4 (A4)*/
/* #include "transfer14inv.h"         File of parameters for case 5  (X29)*/
/* #include "transfer1614hp.h"        File of parameters for case 6 (Optimal)*/
#include "transfer1614hp.h"        /* File of parameters for case 7 (Limited)*/


/* Neuralworks calls the USERIO program through the function UsrIO */

int UsrIO()
        {

/* Declarations */

   extern double ts;               /* Sampling time */
   extern double iterations;       /* number of iterations */
   extern double pow();            /* Power function */
   extern double fmod();           /* Remainder function */
```

```
        extern long random();           /* Random number generator */
        extern char *input_name[];      /* Names of inputs */
        extern double num1[num_in][num_out][ord1];/* Numerator coefficients   */
        extern double den1[ord1];               /* Denominator coefficients  */
        extern double num2[num_in2][num_out2][ord2];/* Numerator coefficients */
        extern double den2[ord2];               /* Denominator coefficients  */
        extern double alpha1;           /* Place holder for Command[]*/
        extern double alpha2;           /* Place holder for Command[]*/
        extern double alpha3;           /* Place holder for Command[]*/

        static int profile={0};
        static int redraw,in={0};       /* Redisplay initialization flag */
        static double check1;           /* Check flag */
        static double check2;           /* Check flag */
        static double count={0.0};      /* Display counter */
        static int input;               /* Selected input */


        static double rmem,rmem1,rmem2;        /* Counters for inputs */
        static double rcount,rcount1,rcount2;  /* Counters for inputs */
                int h,i,j,imax;         /* Indices */
                char buf[90];           /* Display buffer */
                char *sp;               /* String pointer */


/* Feedback regression vectors */

        static double feedback[num_feed]={0.0};
        static double feedback2[num_feed2]={0.0};

/*Ref input + regression vectors */

        static double command[num_comd]= {0.0};

       static double command2[num_comd2]= {0.0};

/* Regression vectors applied to NN */

       static double control[num_cont]= {0.0};

       static double control2[num_cont2]= {0.0};

/* Plant responses to regression vectors */

        static double out[num_out]={0.0};
        static double out2[num_out2]={0.0};



/* Definitions */

#define MAXRAND (0x7fffffff)
#define rand random


/* Definitions of Level 1 */


#define feedback_lay 0
#define command_lay 1
#define control_lay 2
                            /* #define hidden1_lay 3  For non-linear models */
                            /* #define hidden2_lay 4                        */
#define out_lay 3
```

```
/* Definitions of Level 2 */

#define feedback2_lay 4
#define command2_lay 5
#define control2_lay 6
#define out2_lay 7


/* initialization (if necessary) */

IORTNCDE = 0;

        switch ( IOREQCDE ) {


/* ------------------------------------------------------------------------

        case RQ_ATTENTION:

/* User select input to be used */

        Again3:
        sprintf( buf,"\nEnter Desired Input Type (1. %s, 2. %s, 3. %s, 4. %s"
                input_name[1],input_name[2],input_name[3]);
        PutStr( buf );
        sp=GetStr();
        sscanf( sp, "%ld", &input);
        if( input >3. || input<1. ){
                sprintf(  buf, "\n%s",input_name[0] );
                PutStr( buf );
                for(i=0;i<1000;i++){
                }
                goto Again3;
        }


/* Display selections */

        sprintf( buf,"\nInput:  %s  selected",
                input_name[input]);
        PutStr( buf );
        if(input==2. || input==3.){
                PutStr("\nEnsure LR is set to zero for test");
        }
        in=1;
        break;


        case RQ_REWIND:

 /* Occurs at the start of a "learn all".
  * Rewind any input files to the beginning. */

        count=0.0;
        break;


        case RQ_LSTART:

  /* Learn start; occurs once at the start    */
```

```
/* initialize  input if not already done so */

        if(in==0){
                input=1;
                in=1;

        }

/* check if user wishes to redisplay after every plot reaches the end */

        PutStr("\nHow often do you wish to redraw the screen (0 for never)?");
        sp=GetStr();
        sscanf( sp, "%ld", &redraw);


/* start random binary or composite in time sequence */

     if(num_in==1){
        if(input==1){
                rcount=0.0;
                rmem=rand() % 4;
                command[0]=pow(-1.0,rmem);
        }
     }

     if(num_in==2){
        if(input==1){
                rcount1=0.0;
                rcount2=0.0;
                rmem1=rand() % 4;
                rmem2=rand() % 4;
                command[0]=pow(-1.0,rmem1);
                command[1]=pow(-1.0,rmem2);
                if(c==6 || c==7){
                    command2[0]=pow(-1.0,rmem1);
                    command2[1]=pow(-1.0,rmem2);
                }
        }
     }


/* start test with a pulse of 1 degree for 1 sec */

     if(num_in==1){
        if(input==2){
                rcount=50.0;
                command[0]=alpha1;
        }
     }

     if(num_in==2){
        if(input==2){
                rcount1=50.0;
                rcount2=50.0;
                command[0]=alpha1;
                command[1]=0.0;
                if(c==6 || c==7){
                    command2[0]=alpha3;
                    command2[1]=0.0;
                }
          }
```

```
   if(input==3){

           rcount1=50.0;
           rcount2=50.0;
           command[1]=alpha1;
           command[0]=0.0;
           if(c==6 || c==7){
               command2[1]=alpha3;
               command2[0]=0.0;
           }
     }
}


/* display the starting conditions */

        sprintf( buf, "\nCycles:  %f  Input:  %s",
         count, input_name[input]);
        PutStr( buf );
        break;


/*-------------------------------------------------------------------------

        case RQ_LEARNIN:

/* The input values should be stored in IODATA array  */


/* input feedback layers to the network */

        if( IOLAYER==feedback_lay && IOCOUNT==num_feed ){
                for( i=0; i<num_feed;i++ ){
                        IODATA[i]=feedback[i];

                }
        }

        if( IOLAYER==feedback2_lay && IOCOUNT==num_feed2 ){
                for( i=0; i<num_feed2;i++ ){
                        IODATA[i]=feedback2[i];

                }
        }

/* input commmand layers to the network */

        if( IOLAYER==command_lay && IOCOUNT==num_comd){
                for(i=0;i<num_comd;i++){
                        IODATA[i]=command[i];
                }
        }

        if( IOLAYER==command2_lay && IOCOUNT==num_comd2){
                for(i=0;i<num_comd2;i++){
                        IODATA[i]=command2[i];
                }
        }

        break;
```

```
        case RQ_WRSTEP:

/* output control layer from network */

        break;

/*-------------------------------------------------------------------------

        case RQ_LEARNOUT:

/* present plant or model responses to the network */

        if( IOLAYER==out_lay && IOCOUNT==num_out){
                for(i=0;i<num_out;i++){
                IODATA[i]=out[i];
                }
        }

        if( IOLAYER==out2_lay && IOCOUNT==num_out2){
                for(i=0;i<num_out2;i++){
                IODATA[i]=out2[i];
                }
        }
        break;

/*-------------------------------------------------------------------------

        case RQ_LEARNRSLT:

/* control outputs from network */

        if( IOLAYER==control_lay && IOCOUNT==num_in){

                if(num_in==1){
                     for(i=0;i<num_cont;i++){
                          control[i]=command[i];
                     }
                     control[0]=IODATA[0];
                }

                if(num_in==2){
                   imax=ord1;
                   if (num_out==1){
                        imax=reg_vec1;
                   }
                   for (i=0;i<imax;i++){
                        control[i]=command[i+num_out-1];
                   }
                   for (i=2;i<num_in+1;i++){
                        imax=ord1*i;
                        if (num_out==i){
                              imax=reg_vec1;
                        }
                        if (num_out>i-1){
                              for (j=ord1*(i-1)+1;j<imax;j++){
                                    control[j]=command[j+num_out-i];
                              }
                        }
                   }
                   for (i=0;i<num_in;i++){
                   control[ord1*i]=IODATA[i];
                   }
                }
```

```
/* generate system and model response to this control input */

                for(i=0;i<num_out;i++){
                      out[i]=0.00;
                      for(h=0;h<num_in;h++){
                         for(j=0;j<ord1;j++){
                  out[i]=out[i]+num1[h][i][j]*(control[ord1*h+j]);
                         }
                      }
                      for(j=0;j<ord1;j++){
                  out[i]=out[i]+den1[j]*(control[ord1*(i+num_in)+j]);
                      }
                }

        }


        if(IOCOUNT==num_out && IOLAYER==out_lay){

 /* shift the regression vectors */

           if(num_in==1){
                for(i=0;i<num_feed;i++){
                        feedback[i]=control[i];
                }
                for (i=0;i<num_feed;i++){
                        command[i+1]=control[i];
                }
           }

           if(num_in>1){
                for (i=1;i<num_out+1;i++){
                        imax=ord1*i-1;
                        if (num_out==i){
                               imax=reg_vec1-1;
                        }
                        if (num_out>i-1){
                               for (j=ord1*(i-1);j<imax;j++){
                                     feedback[j-(i-1)]=control[j];
                               command[j+num_out-(i-1)]=control[j];
                               }
                        }
                 }
           }


 /* generate a new random binary input */

             if(num_in==1){
                if(input==1){
                        rcount++;
                        command[0]=alpha1;
                        if(fmod(count,2.0)<1.0){
                                rmem=rand() % 4;
                                command[0]=pow(-1.0,rmem);
                                alpha1=command[0];
                        }
                }
              }
```

```
            if(num_in==2){
                if(input==1){
                        rcount1++;
                        command[0]=alpha1;
                        command[1]=alpha2;
                        if(fmod(count,2.0)<1.0){
                                rmem1=rand() % 4;
                                rmem2=rand() % 4;
                        command[0]=pow(-1.0,rmem1);
                        command[1]=pow(-1.0,rmem2);
                                alpha1=command[0];
                                alpha2=command[1];

                         }
                }
            }


/* generate a new test (pulse of 1 degree for 1 sec) input */

            if(num_in==1){
                if(input==2){
                    rcount --;
                    command[0]=alpha1;
                    rmem=200;
                    if(rcount<=0.0){
                       rmem--;
                       command[0]=0.0;
                       if(rmem<=0.0){
                           rcount=50.0;
                           alpha1=1.0;
                       }
                    }
                }
            }

            if(num_in==2){
               if(input==2){
                    rcount1 --;
                    command[0]=alpha1;
                    command[1]=0.0;
                    rmem1=200;
                    if(rcount1<=0.0){
                       rmem1--;
                       command[0]=0.0;
                       if(rmem1<=0.0){
                           rcount1=50.0;
                           alpha1=1.0;
                       }
                    }
                }
                if(input==3){
                    rcount1 --;
                    command[1]=alpha1;
                    command[0]=0.0;
                    rmem1=200;
                    if(rcount1<=0.0){
                       rmem1--;
                       command[1]=0.0;
                       if(rmem1<=0.0){
                           rcount1=50.0;
                           alpha1=1.0;
```

```
                    }
                }
            }

        }


/* load the regressors with system and model responses */

            for(i=0;i<num_out;i++){
                    command[ord1*(i+num_in)]=-out[i];
                    feedback[ord1*(i+num_in)-num_in]=-out[i];
            }


/* load the input of Level 1 to the output of Level 2 */

        if(c==3 || c==4 || c==5){
                out2[0]=command[0];
        }

/* load the outputs of Level 1 to the inputs of Level 2 */

        if(c==3 || c==4 || c==5 || c==8 || ){
               for(i=0;i<num_out;i++){
                    command2[i]=out[i];
            }
        }


/* increment the counter and update displays as necessary */

            count++;
            check1=fmod(count,1000.);

            if(check1<1.0){
            sprintf( buf, "\nCycles:  %f  Input:  %s",
                     count,input_name[input]);
                    PutStr( buf );

       if(c==5 || c==6 || c==7){
          if(input==1){
            for(i=0;i<num_out;i++){
                    if(out[i]>1.0 || out[i]<-1.0){
                            for (i=0;i<num_feed;i++){
                                    feedback[i]=0.0;
                                    command[i+num_in]=0.0;
                            }

                    }
            }
          }
       }
            if(redraw !=0){

                        if(fmod(count,(double)redraw)<1.0){
                                IORTNCDE=1;
                        }
                }
          }
```

```
/*   ONLY FOR CASES 3 TO 8    */


        if( IOLAYER==control2_lay && IOCOUNT==num_in2 ){

                imax=ord2;
                if (num_in2==1){
                        imax=reg_vec2;
                }
                for (i=0;i<imax;i++){
                        control2[i]=command2[i+num_in2-1];
                }
                for (i=2;i<num_in2+1;i++){
                        imax=ord2*i;
                        if (num_in2==i){
                              imax=reg_vec2;
                        }
                        if (num_in2>i-1){
                              for (j=ord2*(i-1)+1;j<imax;j++){
                                    control2[j]=command2[j+num_in2-i];
                              }
                        }
                 }
                 for (i=0;i<num_in2;i++){
                 control2[ord2*i]=IODATA[i];
                 }

/* generate system and model response to this control2 inputs */

           if(c==6  || c==7 || c==8){
                for(i=0;i<num_out2;i++){
                        out2[i]=0.00;
                        for(h=0;h<num_in;h++){
                           for(j=0;j<ord2;j++){
                    out2[i]=out2[i]+num2[h][i][j]*(control2[ord2*h+j]);
                           }
                        }
                        for(j=0;j<ord2;j++){
                    out2[i]=out2[i]+den2[j]*(control2[ord2*(i+2)+j]);
                        }
                 }
            }
        }


        if(IOCOUNT==num_out2 && IOLAYER==out2_lay){

/* shift the regression vectors */

                for (i=1;i<num_out+1;i++){
                        imax=ord2*i-1;
                        if (num_out==i){
                               imax=reg_vec2-1;
                        }
                        if (num_out>i-1){
                               for (j=ord2*(i-1);j<imax;j++){

                                     feedback2[j-(i-1)]=control2[j];
                               command2[j+num_out-(i-1)]=control2[j];
                               }
                        }
                 }
```

```
/* generate a new random binary input */

            if(c==6 || c==7){
                if(input==1){
                        rcount2++;
                        command2[0]=alpha1;
                        command2[1]=alpha2;
                        if(fmod(count,2.0)<1.0){
                                rmem1=rand() % 4;
                                rmem2=rand() % 4;
                        command2[0]=pow(-1.0,rmem1);
                        command2[1]=pow(-1.0,rmem2);
                                alpha1=command2[0];
                                alpha2=command2[1];
                        }
                }
            }


/* generate a new test (pulse of 1 degree for 1 sec) input */

            if(c==6 || c==7){
                if(input==2){
                   rcount2 --;
                   command2[0]=alpha3;
                   command2[1]=0.0;
                   rmem2=200;
                   if(rcount2<=0.0){
                      rmem2--;
                      command2[0]=0.0;
                      if(rmem2<=0.0){
                          rcount2=50.0;
                          alpha3=0.001;
                      }
                   }
                 }
                 if(input==3){
                   rcount2 --;
                   command2[1]=alpha3;
                   command2[0]=0.0;
                   rmem2=200;
                   if(rcount2<=0.0){
                      rmem2--;
                      command2[1]=0.0;
                      if(rmem2<=0.0){
                          rcount2=50.0;
                          alpha3=0.001;
                      }
                   }
                 }
             }

/* load the regressors with system and model responses */

             if(c==3 || c==4 || c==5){
                   command2[ord1*num_out]=-out2[0];
                   feedback2[(ord1-1)*num_out]=-out2[0];
             }

             if(c==6 || c==7 || c==8){
                 for(i=0;i<num_out2;i++){
                       command2[ord2*(i+2)]=-out2[i];
                       feedback2[ord2*(i+2)-2]=-out2[i];
                 }
             }
```

```
/* calculate the error value going into the controller (feedback loop) */

            if(c==8){
                  for (i=0;i<num_out2;i++){
                      command[i]=command[i]-out2[i];
                  }
            }


/* increment the counter and update displays as necessary */

           if(c==6 || c==7){
              if(input==1){
                 for(i=0;i<num_out2;i++){
                       if(out2[i]>1.0 || out2[i]<-1.0){
                               for (i=0;i<num_feed2;i++){
                                               feedback2[i]=0.0;
                                               command2[i+num_in]=0.0;
                               }

                       }

                 }
              }
           }
        }


        break;


/*-------------------------------------------------------------------------

        case RQ_LEND:

/* end learning mode, display current status */

        sprintf( buf, "\nCycles:  %f  Input:  %s",
                        count, input_name[input]);
                       PutStr( buf );

        break;


        case RQ_RSTART:


        break;


        case RQ_READ:

        break;


        case RQ_WRITE:

        break;
```

```
        case RQ_RCLTST:

        break;


        case RQ_REND:

/* end recall , display current status */

        sprintf( buf, "\nCycles:  %f  Input:  %s",
                        count, input_name[input]);
                       PutStr( buf );


        break;

        case RQ_TERM:

        /* terminate userio */

        sprintf( buf, "\nCycles:  %f",
                       count);
                       PutStr( buf );

        break;

        }

        return;
}
```

APPENDIX B: NEURALWORKS PROFESSIONAL II CONTROL STRATEGIES FILES

```
csv2.1
!file format is Control Strategy Version 2.1
!
!
!  Source:       contstr1.nnc
!  Executable:   neuralworks professional II
!  Version:      2
!  Date:         30 Aug 1991
!  Author:       D. J. Collins
!  Co-Author     D. J. Bertrand
!  Project:      Neural Networks in Adaptive Control
!  Environment:  UNIX/SunOS/Neuralworks Control Strategy
!  Path:         eileen:/fil669/home/monika
!  Description:  This is a prototype control strategy for use with
!                the simomonika USERIO program.
!                This strategy uses a proprietary language which is covered
!                in some detail in the Neuralworks Professional II manual.
!                This control strategy applies only for cases 1, 2, 3, 4 & 5
!                which require Level 1 to be trained first.
!                Each network must use the back-propagation learning concept, and
!                possesses at least an input or feedback layer, a control or
!                command layer, and an output layer.
!
!
!
!  Revisions:    Handles all hidden layers
!
!
!MASK    label  op-code operands        comment
L_saRisa        trace   aux3            !   set trace option to aux3
Li_aRisa        cset    recall,0        !   recall count
!
! Get inputs for Level 1 (learn and recall)
!
L_saRisa        lset    in              !   set feedback layer or input
L_saR___        io      lrnin           !   get feedback vector learn
L___Risa        io      read            !   get feedback vector recall
L_saRisa        lset    cur,1           !   set command layer
L_saR___        io      lrnin           !   get command vector learn
L___Risa        io      read            !   get command vector recall
!
!  Start forward pass to but not including output layer
!

L_saRisa        lset    in              !   set feedback layer or input
L_saR___ @loop1 math    sum|lnoise|tran|output|e=0|fire ! fire 0th layer learn
L___Risa        math    sum|rnoise|tran|output|e=0  !   fire 0th layer recall
L_saRisa        lset    cur,1           !   set next layer  learn & recall
L_saRisa        lcmp    in,+3           ! at output layer ?
L_saRisa        blt     @loop1          ! loop till done
!
! Transfer control vector to userio and get desired output
!
L_saRisa        lset    in,+2           ! set control layer learn and recall
L_saR___        io      lrnrslt         ! sent control input   learn
L___Risa        io      write           ! sent control input   recall
L_saRisa        lset    in,+3           ! set output layer  learn and recall
L_saR___        io      lrnout          ! get output layer desired learn
L___Risa        io      rcltst          ! get output layer desired  recall
!
```

```
! Compute final outputs at output layer
!
L_saR___          math    sum|lnoise|ce=e|tran|output|e-=w|fire|e*=ef|swap|fire !
L___Risa          math    sum|lnoise|ce=e|tran|output|e-=w|swap   !recall
L___Risa          eos
!
!  write results to userio
!
L_saR___          io      lrnrslt          ! sent output    learn
L___Risa          io      write            ! sent output   recall
!
! learn cycle  back propagate error  (note at output layer)
!
L_saR___          math    i=e|e=ce         ! put desire value in sum field
L_saR___ @lloop   math    ce=e|e*=f'|backp|learn|fire  !   bkp/learn 3rd
L_saR___          lset    cur,-1           !   previous layer
L_saR___          lcmp    in               !   at input layer?
L_saR___          bgt     @lloop           !   loop till done
L__aRisa          trace   0                !   turn off any trace function
!
! Note, when viewing an output node, output contains the network result,
! sum contains the desired result, and ce contains the error.
```

```
csv2.1
!file format is Control Strategy Version 2.1
!
!
!  Source:       contstr2.nnc
!  Executable:   neuralworks professional II
!  Version:      2
!  Date:         30 Aug 1991
!  Author:       D. J. Bertrand
!  Project:      Neural Networks in Adaptive Control
!  Environment:  UNIX/SunOS/Neuralworks Control Strategy
!  Path:         eileen:/fil669/home/Monika
!  Description:  This is a prototype control strategy for use with
!                SimoMonika.c, the USERIO program.
!                This strategy uses a proprietary language which is covered
!                in some detail in the Neuralworks Professional II manual.
!                This program is a continuity to contstra1.nnc.
!
!                For cases 3,4 & 5 :
!                     After Level 1 has been trained with contstra1,
!                     Level 2 is then trained with this control strategy using
!                     the inputs and outputs of Level 1 as references.
!
!                For cases 6 & 7 :
!                     Contstra1 is not necessary, since both Level 1 & 2 can
!                     be emulated simultaneously with contstra2.
!
!                For case 8 :
!                     No training takes place. Make sure LR is set
!                     to zero before using contstra2.
!
!
!
!MASK    label  op-code operands        comment
L_saRisa        trace   aux3            !   set trace option to aux3
Li_aRisa        cset    recall,0        !   recall count
!
! Get inputs  of Level 1 (learn and recall)
!
L_saRisa        lset    in              !   set feedback layer or input
L_saR___        io      lrnin           !   get feedback vector learn
L___Risa        io      read            !   get feedback vector recall
L_saRisa        lset    cur,1           !   set command layer
L_saR___        io      lrnin           !   get command vector learn
L___Risa        io      read            !   get command vector recall
!
!  Start forward pass to but not including output layer
!
L_saRisa        lset    in              !   set feedback layer or input
L_saR___ @loop1 math    sum|lnoise|tran|output|e=0|fire !fire 0th layer learn
L___Risa        math    sum|rnoise|tran|output|e=0  !   fire 0th layer recall
L_saRisa        lset    cur,1           !   set next layer  learn & recall
L_saRisa        lcmp    in,+3           ! at output layer ?
L_saRisa        blt     @loop1          ! loop till done
!
! Transfer control vector to userio and get desired output
!
L_saRisa        lset    in,+2           ! set control layer learn and recall
L_saR___        io      lrnrslt         ! sent control input   learn
L___Risa        io      write           ! sent control input   recall
```

```
L_saRisa        lset    in,+3           ! set output layer  learn and recall
L_saR___        io      lrnout          ! get output layer desired learn
L___Risa        io      rcltst          ! get output layer desired  recall
!
! Compute final outputs at output layer
!
L_saR___        math    sum|lnoise|ce=e|tran|output|e-=w|fire|e*=ef|swap|fire !
L___Risa        math    sum|lnoise|ce=e|tran|output|e-=w|swap   !recall
L___Risa        eos
!
!  write results to userio
!
L_saR___        io      lrnrslt         ! sent plant output    learn
L___Risa        io      write           ! sent plant output   recall
!
! learn cycle  back propagate error  (located at output layer)
!
L_saR___        math    i=e|e=ce        ! put desire value in sum field
L_saR___ @loop2 math    ce=e|e*=f'|backp|learn|fire  !   bkp/learn 3rd
L_saR___        lset    cur,-1          !   previous layer
L_saR___        lcmp    in              !   at input layer?
L_saR___        bgt     @loop2          !   loop till done
!
! note that the error does not need to be backpropagated if using cases 2 & 3.
! It has already been trained using estimat1.nnc.
!
!
! Get inputs of Level 2 (learn and recall)
!
L_saRisa        lset    in,+4           !   set feedback2 layer or input
L_saR___        io      lrnin           !   get feedback2 vector learn
L___Risa        io      read            !   get feedback2 vector recall
L_saRisa        lset    cur,1           !   set command2 layer
L_saR___        io      lrnin           !   get command2 vector learn
L___Risa        io      read            !   get command2 vector recall
!
!  Start forward pass to but not including output2 layer
!
!
L_saRisa        lset    in,+4           !   set feedback2 layer or input
L_saR___ @loop3 math    sum|lnoise|tran|output|e=0|fire !fire 0th layer learn
L___Risa        math    sum|rnoise|tran|output|e=0  !   fire 0th layer recall
L_saRisa        lset    cur,1           !   set next layer  learn & recall
L_saRisa        lcmp    in,+7           ! at output2 layer ?
L_saRisa        blt     @loop3          ! loop till done
!
! Transfer control2 vector to userio and get desired output2
!
L_saRisa        lset    in,+6           ! set control2 layer learn and recall
L_saR___        io      lrnrslt         ! sent control2 input   learn
L___Risa        io      write           ! sent control2 input   recall
L_saRisa        lset    in,+7           ! set output2 layer  learn and recall
L_saR___        io      lrnout          ! get output2 layer desired learn
L___Risa        io      rcltst          ! get output2 layer desired  recall
!
! Compute final outputs at output2 layer
!
L_saR___        math    sum|lnoise|ce=e|tran|output|e-=w|fire|e*=ef|swap|fire
L___Risa        math    sum|lnoise|ce=e|tran|output|e-=w|swap   !recall
L___Risa        eos
```

```
!
!  write results to userio
!
L_saR___          io      lrnrslt          ! sent output2 learn
L___Risa          io      write            ! sent output2 recall
!
! learn cycle  back propagate error  (note at output2 layer)
!
L_saR___          math    i=e|e=ce         ! put desire value in sum field
L_saR___ @loop4   math    ce=e|e*=f'|backp|learn|fire  !   bkp/learn 3rd
L_saR___          lset    cur,-1           !   previous layer
L_saR___          lcmp    in,+4            !   at input2 layer?
L_saR___          bgt     @loop4           !   loop till done
L__aRisa          trace   0                !   turn off any trace function
!
!
! Note, when viewing an output node, output contains the network result,
! sum contains the desired result, and ce contains the error.
```

APPENDIX C: NEURALWORKS PROFESSIONAL II HEADER FILES

```
/*****************************************************************************
 *  11 april 1991                                                            *
 *  transfer30hp.txt   (Optimal case )                                       *
 *  Header file for transfer function variables                              *
 *  Sampling time 0.02 sec                                                   *
 *  CAPT D. Bertrand                                                         *
*****************************************************************************/

/* Define the case # */

#define   c        1      /* X-29, 30 states closed-loop conf. */


/* Define the variables */

/* Level 1 */
#define ord1      30      /* order of the X-29 closed-loop conf. */
#define num_in     2      /* number of inputs to the X-29 closed-loop conf.*/
#define reg_vec1 120      /* regression vector =ord*(num_in+num_out) */
#define num_feed 118      /* feedback layer =reg_vec-num_in */
#define num_comd 120      /* command layer = reg_vec */
#define num_cont 120      /* control layer = reg_vec */
#define num_out    2      /* output layer = num_out */

/* Level 2    ( not required )  */
#define ord2       1      /* order of the X-29 inverse closed-loop conf.*/
#define num_in2    1      /* number of inputs to the X-29 inverse closed-loop*/
#define reg_vec2   1      /* regression vector =ord*(num_in+num_out) */
#define num_feed2  1      /* feedback2 layer = reg_vec-num_out */
#define num_comd2  1      /* command2 layer= reg_vec */
#define num_cont2  1      /* control2 layer = reg_vec */
#define num_out2   1      /* output2 layer = num_in */


/* Declarations */

static double ts={0.02};
static double alpha1={1.0};
static double alpha2={0.01};
static double alpha3={0.001};

static char *input_name[]={"Illegal Input","Random Binary",
   "A Pulse input of 1 degree for 1 sec (input 1)","(input 2)"};


static char *state_name[]={"Illegal State","u(t)","alpha(t)",
        "q(t)", "theta(t)"};



/* Numerator coefficients of the X-29 closed-loop conf. */
/* Order is a1-a30 & q1-q30 for the indices */

static double num1[2][2][30]=
      {    7.217941790123916e-03,
     4.136127382910360e-02,
    -4.669768974859352e-01,
```

```
 1.740915378910785e+00,
-3.860972079490637e+00,
 5.910026001863514e+00,
-6.603623724905617e+00,
 5.461815191735525e+00,
-3.270158987514236e+00,
 1.276667299594394e+00,
-1.520754074600887e-01,
-2.051159049792766e-01,
 1.945345556135285e-01,
-1.056009840779120e-01,
 4.305024816802305e-02,
-1.421564425160682e-02,
 3.890465813729260e-03,
-8.753878576499303e-04,
 1.567504967623089e-04,
-2.108457015082093e-05,
 1.925419095923840e-06,
-9.276539501321390e-08,
-7.822692189116382e-10,
 3.606658455830145e-10,
-1.731282437642258e-11,
 2.828830223439087e-13,
-1.035513245405491e-15,
 3.084625475376078e-17,
 6.087629110463293e-20,
 2.188724968031335e-33,
 3.030279064248020e-02,
-2.176617363311379e-01,
 7.199418522371559e-01,
-1.445887468831245e+00,
 1.937232438753256e+00,
-1.744010585139847e+00,
 9.046191030770956e-01,
 7.520983838907114e-02,
-6.942919440655828e-01,
 8.180108728891611e-01,
-6.296351231172181e-01,
 3.715234366137601e-01,
-1.779572331145785e-01,
 7.113143215820728e-02,
-2.403378264476541e-02,
 6.879198166074046e-03,
-1.656190902514149e-03,
 3.302644120224971e-04,
-5.332055003518747e-05,
 6.764447617547078e-06,
-6.496114358148548e-07,
 4.502381161607562e-08,
-2.099033629117875e-09,
 5.705913270502097e-11,
-4.936196094124739e-13,
-1.463168035258393e-14,
 3.339921369161159e-16,
-1.652969295184519e-18,
 4.511432668157739e-20,
 1.618497006367023e-31,
 2.913303838345982e-03,
-1.859587162627463e-03,
-7.257319744519464e-02,
```

```
 3.655964413534321e-01,
-9.606361329579727e-01,
 1.693827322488687e+00,
-2.182839335964559e+00,
 2.128533975768278e+00,
-1.587246649450236e+00,
 8.986946032316325e-01,
-3.730810625463050e-01,
 9.989649964975330e-02,
-4.970326697264049e-03,
-1.160762684818817e-02,
 7.783152259177362e-03,
-3.255752858643685e-03,
 1.045405054255093e-03,
-2.683453425099142e-04,
 5.432548388124316e-05,
-8.285696168148078e-06,
 8.778748619323248e-07,
-5.498966408340436e-08,
 1.047635489968687e-09,
 9.124468024653338e-11,
-5.975619276082550e-12,
 1.069871022603523e-13,
-3.507375971593066e-16,
 1.211440893663403e-17,
 2.074970670899496e-20,
 1.496428514248921e-33,
 4.943906040469592e-02,
-2.486659847197288e-01,
 4.696527405284456e-01,
-1.710384843461270e-01,
-1.086534623251680e+00,
 2.936298628985867e+00,
-4.355953352923223e+00,
 4.581740987760242e+00,
-3.705662442825542e+00,
 2.399924940838602e+00,
-1.276938538460257e+00,
 5.681969684923196e-01,
-2.137895540465093e-01,
 6.824380985107714e-02,
-1.836615026668542e-02,
 4.094699081625131e-03,
-7.327299957345171e-04,
 9.966153548749378e-05,
-9.192870090060862e-06,
 3.638980355619663e-07,
 3.811403848043370e-08,
-8.162814699140338e-09,
 7.514384787162991e-10,
-4.201909270278608e-11,
 1.427936059062508e-12,
-2.689202646566666e-14,
 2.645984174551336e-16,
-2.499266947496521e-18,
 1.613683912162787e-20,
 1.105945879965577e-31
        };
```

```
/* Denominator coefficients */


  static double den1[30]=
          {  -8.702573379180253e+00,
       3.624185054971434e+01,
      -9.637476611139380e+01,
       1.840944567247478e+02,
      -2.691932874966105e+02,
       3.133425587022512e+02,
      -2.979853544537900e+02,
       2.358162965122712e+02,
      -1.574535322515088e+02,
       8.967590202531342e+01,
      -4.395124648498849e+01,
       1.866251064997256e+01,
      -6.894463299309602e+00,
       2.218004644230438e+00,
      -6.194736024651941e-01,
       1.490902392482576e-01,
      -3.055808663119203e-02,
       5.250501120595874e-03,
      -7.417915268847094e-04,
       8.426982161281500e-05,
      -7.509227926740248e-06,
       5.105441468225222e-07,
      -2.559001820930592e-08,
       8.981854293991370e-10,
      -2.024422876299269e-11,
       2.556068754189784e-13,
      -1.858391083686945e-15,
       2.070784849258401e-17,
      -5.231469022776601e-21,
      -8.203828579087643e-38
                  };

  /* Does not apply for the case 1 */

  static double num2[1][1][1]= {1.0};

  static double den2[1]={1.0};
```

```
/*****************************************************************************
 *  11 april 1991                                                            *
 *  transfer30lim.txt   (Limited case )                                      *
 *  Header file for transfer function variables                              *
 *  Sampling time 0.02 sec                                                   *
 *  CAPT D. Bertrand                                                         *
*****************************************************************************/

/* Define the case # */

#define   c       2       /* X-29, 30 states closed-loop conf. (limited case)*/


/* Define the variables */

/* Level 1 */
#define ord1      30      /* order of the X-29 closed-loop conf. */
#define num_in     2      /* number of inputs to the X-29 closed-loop conf.*/
#define reg_vec1 120      /* regression vector =ord*(num_in+num_out) */
#define num_feed 118      /* feedback layer =reg_vec-num_in */
#define num_comd 120      /* command layer = reg_vec */
#define num_cont 120      /* control layer = reg_vec */
#define num_out    2      /* output layer = num_out */

/* Level 2       ( not required )  */
#define ord2       1      /* order of the X-29 inverse closed-loop conf.*/
#define num_in2    1      /* number of inputs to the X-29 inverse closed-loop*/
#define reg_vec2   1      /* regression vector =ord*(num_in+num_out) */
#define num_feed2  1      /* feedback2 layer = reg_vec-num_out */
#define num_comd2  1      /* command2 layer= reg_vec */
#define num_cont2  1      /* control2 layer = reg_vec */
#define num_out2   1      /* output2 layer = num_in */


/* Declarations */

static double ts={0.02};
static double alpha1={1.0};
static double alpha2={0.01};
static double alpha3={0.001};

static char *input_name[]={"Illegal Input","Random Binary",
   "A Pulse input of 1 degree for 1 sec (input 1)","(input 2)"};


static char *state_name[]={"Illegal State","u(t)","alpha(t)",
        "q(t)", "theta(t)"};



/* Numerator coefficients of the X-29 closed-loop conf. */
/* Order is a1-a30 & q1-q30 for the indices */

static double num1[2][2][30]=
      {    2.844220109210482e-05,
    -4.636343490460604e-05,
    -4.816337071247290e-04,
     2.541965352179432e-03,
```

```
-6.221344563186904e-03,
 9.700836906404220e-03,
-1.056937401995128e-02,
 8.128098827000940e-03,
-3.979045497317202e-03,
 4.550147278905570e-04,
 1.233249102824630e-03,
-1.384403827046299e-03,
 9.040024800270885e-04,
-4.324739839844938e-04,
 1.615871994147255e-04,
-4.804316744344561e-05,
 1.122824825117164e-05,
-1.943751086891377e-06,
 2.022629447609448e-07,
 3.408939495269658e-09,
-5.704020784890775e-09,
 1.058982467133109e-09,
-9.700269350498115e-11,
 3.954993550955955e-12,
 1.556257997037185e-14,
-6.316655529366355e-15,
 1.434341976227709e-16,
-5.284811098416118e-19,
 1.694663132378100e-20,
 2.280431460264726e-26,
 1.302867254258544e-03,
-7.382851430506321e-03,
 1.133243132534290e-02,
 2.624436184066781e-02,
-1.598754578815260e-01,
 3.871527027772572e-01,
-6.112169441003061e-01,
 7.102948491758525e-01,
-6.420179001261204e-01,
 4.654773480224321e-01,
-2.756585281389334e-01,
 1.345900453700608e-01,
-5.422295001227440e-02,
 1.783262619678538e-02,
-4.631680402436444e-03,
 8.617029016368960e-04,
-6.950853850051786e-05,
-2.247015885420423e-05,
 1.205464699656489e-05,
-3.218590077396812e-06,
 5.913539012463971e-07,
-7.844577461731442e-08,
 7.421834511632813e-09,
-4.815724560938900e-10,
 2.020594289801237e-11,
-5.015798784974100e-13,
 6.438926158022875e-15,
-4.595473491280940e-17,
 5.502338148222649e-19,
 1.714467588949488e-24,
 3.414771400578331e-05,
-1.114930428087746e-04,
-1.841134245808007e-04,
 1.854687459513116e-03,
```

```
-5.586092871340043e-03,
 1.064962476698383e-02,
-1.499862810806007e-02,
 1.648927429573632e-02,
-1.436138159283473e-02,
 9.866396810934930e-03,
-5.258203378502913e-03,
 2.092821291967084e-03,
-5.565361172443772e-04,
 4.807825585118053e-05,
 4.066271012703737e-05,
-2.719419553098845e-05,
 1.033156362439946e-05,
-2.955581602348545e-06,
 6.870538006159554e-07,
-1.316939415157981e-07,
 2.026221445272300e-08,
-2.361233864223112e-09,
 1.921063600537676e-10,
-9.809951611347394e-12,
 2.653279103876279e-13,
-2.319233372880782e-15,
-1.317187061896219e-17,
-2.407012535891711e-19,
-4.433880118258184e-21,
 2.735903505022465e-26,
 1.596981542963505e-03,
-8.528373703732939e-03,
 1.086196106746229e-02,
 3.846897564267238e-02,
-1.976730935505202e-01,
 4.554354535725906e-01,
-7.004572410351102e-01,
 8.039373333258482e-01,
-7.256272931053900e-01,
 5.302621924158757e-01,
-3.189146614438982e-01,
 1.590513902863222e-01,
-6.573301084867467e-02,
 2.225901012112530e-02,
-5.988950029420792e-03,
 1.177842478515601e-03,
-1.182446502181467e-04,
-2.078456649373672e-05,
 1.369232238676577e-05,
-3.801953769720645e-06,
 7.081334352697583e-07,
-9.437638502624414e-08,
 8.936766759591824e-09,
-5.794705377000146e-10,
 2.428464299936500e-11,
-6.021234370560187e-13,
 7.724462474106624e-15,
-5.516671952418855e-17,
 6.595904191359686e-19,
 2.056900519282188e-24
        };
```

```
/* Denominator coefficients */


static double den1[30]=
        {  -1.135087146556161e+01,
     6.093263249866681e+01,
    -2.061911065822222e+02,
     4.948035132731752e+02,
    -8.983923725483406e+02,
     1.286089683000355e+03,
    -1.493744793706265e+03,
     1.437421369904793e+03,
    -1.164296889620834e+03,
     8.034380606787025e+02,
    -4.766521786713293e+02,
     2.447238944329588e+02,
    -1.092086442202805e+02,
     4.244916189022602e+01,
    -1.437006104677757e+01,
     4.225905264868553e+00,
    -1.073931722819647e+00,
     2.338884065807147e-01,
    -4.313167043461474e-02,
     6.624455192814090e-03,
    -8.286628884990565e-04,
     8.195010027770559e-05,
    -6.160373942002916e-06,
     3.345416930770729e-07,
    -1.227921681486361e-08,
     2.775475801830879e-10,
    -3.410287170188014e-12,
     2.517895554215908e-14,
    -2.742185292507811e-16,
     1.276107428420735e-33
               };

/* Does not apply for the case 1 */

static double num2[1][1][1]= {1.0};

static double den2[1]={1.0};
```

```
/*************************************************************************
 *  11 april 1991                                                        *
 *  transferhp1inv.txt   (Optimal case input 1)                          *
 *  Header file for transfer function variables                          *
 *  Sampling time 0.02 sec                                               *
 *  CAPT D. Bertrand                                                     *
*************************************************************************/

/* Define the case # */

#define   c        3      /* X-29 inverse closed-loop conf. structure */


/* Define the variables */

/* Level 1 */
#define ord1      30      /* order of the X-29 closed-loop conf. */
#define num_in     1      /* number of inputs to the X-29 closed-loop conf.*/
#define reg_vec1  90      /* regression vector =ord*(num_in+num_out) */
#define num_feed  89      /* feedback layer =reg_vec-num_in */
#define num_comd  90      /* command layer = reg_vec */
#define num_cont  90      /* control layer = reg_vec */
#define num_out    2      /* output layer = num_out */

/* Level 2 */
#define ord2      30      /* order of the X-29 inverse closed-loop conf.*/
#define num_in2    2      /* number of inputs to the X-29 inverse closed-loop*/
#define reg_vec2  90      /* regression vector =ord*(num_in+num_out) */
#define num_feed2 88      /* feedback2 layer = reg_vec-num_out */
#define num_comd2 90      /* command2 layer= reg_vec */
#define num_cont2 90      /* control2 layer = reg_vec */
#define num_out2   1      /* output2 layer = num_in */


/* Declarations */

static double ts={0.02};
static double alpha1={1.0};
static double alpha2={0.01};
static double alpha3={0.001};

static char *input_name[]={"Illegal Input","Random Binary",
   "A Pulse input of 1 degree for 1 sec (input 1)","(input 2)"};


static char *state_name[]={"Illegal State","u(t)","alpha(t)",
        "q(t)", "theta(t)"};


/* Numerator coefficients of the X-29 closed-loop conf. */
/* Order is a1-a30 & q1-q30 for the indices */

static double num1[1][2][30]=
      {    7.217941790123916e-03,
     4.136127382910360e-02,
    -4.669768974859352e-01,
     1.740915378910785e+00,
    -3.860972079490637e+00,
```

```
 5.910026001863514e+00,
-6.603623724905617e+00,
 5.461815191735525e+00,
-3.270158987514236e+00,
 1.276667299594394e+00,
-1.520754074600887e-01,
-2.051159049792766e-01,
 1.945345556135285e-01,
-1.056009840779120e-01,
 4.305024816802305e-02,
-1.421564425160682e-02,
 3.890465813729260e-03,
-8.753878576499303e-04,
 1.567504967623089e-04,
-2.108457015082093e-05,
 1.925419095923840e-06,
-9.276539501321390e-08,
-7.822692189116382e-10,
 3.606658455830145e-10,
-1.731282437642258e-11,
 2.828830223439087e-13,
-1.035513245405491e-15,
 3.084625475376078e-17,
 6.087629110463293e-20,
 2.188724968031335e-33,
 3.030279064248020e-02,
-2.176617363311379e-01,
 7.199418522371559e-01,
-1.445887468831245e+00,
 1.937232438753256e+00,
-1.744010585139847e+00,
 9.046191030770956e-01,
 7.520983838907114e-02,
-6.942919440655828e-01,
 8.180108728891611e-01,
-6.296351231172181e-01,
 3.715234366137601e-01,
-1.779572331145785e-01,
 7.113143215820728e-02,
-2.403378264476541e-02,
 6.879198166074046e-03,
-1.656190902514149e-03,
 3.302644120224971e-04,
-5.332055003518747e-05,
 6.764447617547078e-06,
-6.496114358148548e-07,
 4.502381161607562e-08,
-2.099033629117875e-09,
 5.705913270502097e-11,
-4.936196094124739e-13,
-1.463168035258393e-14,
 3.339921369161159e-16,
-1.652969295184519e-18,
 4.511432668157739e-20,
 1.618497006367023e-31
   };
```

```
/* Denominator coefficients */

static double den1[30]=

        {  -8.702573379180253e+00,
     3.624185054971434e+01,
    -9.637476611139380e+01,
     1.840944567247478e+02,
    -2.691932874966105e+02,
     3.133425587022512e+02,
    -2.979853544537900e+02,
     2.358162965122712e+02,
    -1.574535322515088e+02,
     8.967590202531342e+01,
    -4.395124648498849e+01,
     1.866251064997256e+01,
    -6.894463299309602e+00,
     2.218004644230438e+00,
    -6.194736024651941e-01,
     1.490902392482576e-01,
    -3.055808663119203e-02,
     5.250501120595874e-03,
    -7.417915268847094e-04,
     8.426982161281500e-05,
    -7.509227926740248e-06,
     5.105441468225222e-07,
    -2.559001820930592e-08,
     8.981854293991370e-10,
    -2.024422876299269e-11,
     2.556068754189784e-13,
    -1.858391083686945e-15,
     2.070784849258401e-17,
    -5.231469022776601e-21,
    -8.203828579087643e-38
                };

/* Does not apply for the inverse plant */

static double num2[2][1][30]= {1.0};

static double den2[30]={1.0};
```

```
/*****************************************************************************
 *  6 Sept 1991                                                              *
 *  transferA4inv.txt                                                        *
 *  Header file for transfer function variables                              *
 *  Sampling time 0.1 sec                                                    *
 *  Capt D. Bertrand                                                         *
 *****************************************************************************/


/* Define the case # */

#define   c       4       /* A4 inverse plant structure */


/* Define the variables */

/* Level 1 */
#define ord1       4      /* order of the A4 plant */
#define num_in     1      /* number of inputs to the A4 plant */
#define reg_vec1  20      /* regression vector =ord*(num_in+num_out) */
#define num_feed  19      /* feedback layer =reg_vec-num_in */
#define num_comd  20      /* command layer = reg_vec */
#define num_cont  20      /* control layer = reg_vec */
#define num_out    4      /* output layer = num_out */

/* Level 2 */
#define ord2       4      /* order of the A4 inverse plant */
#define num_in2    4      /* number of inputs to the A4 inverse plant */
#define reg_vec2  20      /* regression vector =ord*(num_in+num_out) */
#define num_feed2 16      /* feedback2 layer = reg_vec-num_out */
#define num_comd2 20      /* command2 layer= reg_vec */
#define num_cont2 20      /* control2 layer = reg_vec */
#define num_out2   1      /* output2 layer = num_in */


/* Declarations */

static double ts={0.1};
static double alpha1={1.0};
static double alpha2={1.0};
static double alpha3={0.001};


static char *input_name[]={"Illegal Input","Random Binary",
   "A Pulse input of 1 degree for 1 sec (input 1)","(input 2)"};


static char *state_name[]={"Illegal State","u(t)","alpha(t)",
        "q(t)", "theta(t)"};


/* Numerator coefficients for the A4 plant*/
/* Order is u1-u4, a1-a4, q1-q4 & t1-t4 for the indices */

static double num1[1][4][4]=
        {
        2.713068210091762e-05,
```

```
     7.724759756833066e-05,
    -7.055791100896158e-05,
    -2.257937441507707e-05,
    -3.461928651266444e-02,
     4.502235817679079e-02,
     1.371511259671188e-02,
    -2.412412258081098e-02,
    -1.986449763497378e-01,
     5.800247578329634e-01,
    -5.641366908427741e-01,
     1.827569093595484e-01,
    -7.705180437452608e-03,
     7.478124872434933e-03,
     6.964010411101729e-03,
    -6.738625984288427e-03
                };

/* Denominator coefficients for the A4 plant */

static double den1[4]=
        {
         -3.694923643854822e+00,
     5.180217304754828e+00,
    -3.275499735648201e+00,
     7.902148612567799e-01
        };


/* Does not apply for the inverse plant */

static double num2[4][1][4]= {0.0};

static double den2[4]={0.0};
```

```
/*****************************************************************************
 *  28 july 1991                                                             *
 *  transfer14inv.txt                                                        *
 *  Header file for transfer function variables                              *
 *  Sampling time 0.02 sec                                                   *
 *  CAPT D. Bertrand                                                         *
 *                                                                           *
*****************************************************************************/

/* Define the case # */

#define   c       5       /* X-29 Inverse plant structure */


/* Define the variables */

/* Level 1 */
#define ord1      14      /* order of the X-29 Plant */
#define num_in     1      /* number of inputs to the X-29 Plant */
#define reg_vec1  42      /* regression vector =ord1*(num_in+num_out) */
#define num_feed  41      /* feedback layer =reg_vec-num_in */
#define num_comd  42      /* command layer = reg_vec */
#define num_cont  42      /* control layer = reg_vec */
#define num_out    2      /* output layer = num_out */

/* Level 2 */
#define ord2      14      /* order of the X-29 inverse plant */
#define num_in2    2      /* number of inputs to the inverse plant */
#define reg_vec2  42      /* regression vector =ord2*(num_in+num_out) */
#define num_feed2 40      /* feedback2 layer = reg_vec-num_out */
#define num_comd2 42      /* command2 layer= reg_vec */
#define num_cont2 42      /* control2 layer = reg_vec */
#define num_out2   1      /* output2 layer = num_in */


/* Declarations */

static double ts={0.02};
static double alpha1={1.0};
static double alpha2={0.01};
static double alpha3={0.001};


static char *input_name[]={"Illegal Input","Random Binary",
   "A Pulse input of 1 degree for 1 sec (input 1)","(input 2)"};

static char *state_name[]={"Illegal State","alpha(t)",
        "q(t)"};



/* Numerator coefficients for the X-29 plant */


static double num1[1][2][14]=
      {   -4.056931727527413e-04,
     8.640985801235956e-04,
    -9.645374778415317e-04,
```

```
     7.668732134167300e-04,
    -3.355203324524325e-04,
     2.002988600113831e-05,
     4.423899844696599e-05,
    -2.134656278755809e-05,
     4.206573556321883e-06,
    -2.414898471374619e-07,
     3.935541586379038e-09,
    -1.296789423146734e-09,
     6.012915966444080e-11,
     6.139928164096763e-13,
    -1.468836867932399e-02,
     5.067239044670480e-02,
    -8.185463286921646e-02,
     8.351205855023736e-02,
    -5.856919312782694e-02,
     2.927929467703660e-02,
    -1.082648765944083e-02,
     2.950804399305770e-03,
    -5.601566387111508e-04,
     6.631958214738630e-05,
    -2.050337748895240e-06,
    -1.175920855414365e-07,
     4.286644733642849e-09,
     6.953985652098497e-11
           };

/* Denominator coefficients for the plant */

static double den1[14]=
        { -4.729739487034623e+00,
     9.756583616420810e+00,
    -1.169645031172698e+01,
     9.179964647228649e+00,
    -5.025697420203761e+00,
     1.978913683674686e+00,
    -5.630838727994958e-01,
     1.137480768724702e-01,
    -1.554252095215423e-02,
     1.300296942834350e-03,
    -5.737414053144769e-05,
     1.055438679886130e-06,
    -4.924500562373306e-09,
     1.114762000808644e-10
           };

/* Does not apply for the inverse plant */

static double num2[2][1][14]= {1.0};

static double den2[14]={1.0};
```

```
/*****************************************************************************
 *  28 july 1991                                                             *
 *  transfer1614hp.txt       OPTIMAL CASE                                    *
 *  Header file for transfer function variables                              *
 *  Sampling time 0.02 sec                                                   *
 *  CAPT D. Bertrand                                                         *
 *                                                                           *
*****************************************************************************/

/* Define the case # */

#define   c        6      /* controller and plant in series (optimal case) */


/* Define the variables */

/* Level 1 */
#define ord1      16      /* order of the controller */
#define num_in     2      /* number of inputs to the controller */
#define reg_vec1  64      /* regression vector =ord1*(num_in+num_out) */
#define num_feed  62      /* feedback layer =reg_vec-num_in */
#define num_comd  64      /* command layer = reg_vec */
#define num_cont  64      /* control layer = reg_vec */
#define num_out    2      /* output layer = num_out */

/* Level 2 */
#define ord2      14      /* order of the plant */
#define num_in2    2      /* number of inputs to the plant */
#define reg_vec2  56      /* regression vector =ord2*(num_in+num_out) */
#define num_feed2 54      /* feedback2 layer = reg_vec-num_out */
#define num_comd2 56      /* command2 layer= reg_vec */
#define num_cont2 56      /* control2 layer = reg_vec */
#define num_out2   2      /* output2 layer = num_in */


/* Declarations */

static double ts={0.02};
static double alpha1={1.0};
static double alpha2={1.0};
static double alpha3={0.001};


static char *input_name[]={"Illegal Input","Random Binary",
   "A Pulse input of 1 degree for 1 sec (input 1)","(input 2)"};

static char *state_name[]={"Illegal State","alpha(t)",
        "q(t)"};



/* Numerator coefficients for the controller */


static double num1[2][2][16]=
      {  -9.859880629593550e+01,
     4.651873138146647e+02,
    -1.051882105730881e+03,
```

```
 1.519154936044425e+03,
-1.557965738284664e+03,
 1.211042887906508e+03,
-7.496709549954828e+02,
 3.744973736643639e+02,
-1.478541553785084e+02,
 4.466434156470294e+01,
-1.003081522244191e+01,
 1.620404514735607e+00,
-1.742798572922879e-01,
 9.724575442910928e-03,
-2.035547071170431e-04,
 8.745388073736235e-13,
-4.388188326607877e+01,
-4.616177234468056e+01,
 6.392111145682509e+02,
-1.397448977654542e+03,
 1.523867810494335e+03,
-9.979620575051404e+02,
 4.152396606507909e+02,
-1.061478517036687e+02,
 1.269598903999647e+01,
 1.194080535740361e+00,
-7.055549558186406e-01,
 1.056283616703849e-01,
-6.324466886079779e-03,
 7.096463377705847e-05,
 3.281163826433534e-06,
 3.825564625565415e-11,
 3.792362230268161e+01,
-1.803218187444186e+02,
 3.809935405874783e+02,
-4.826993636192602e+02,
 4.091641992932675e+02,
-2.334446181168790e+02,
 7.988665887868338e+01,
-7.028001865307544e+00,
-8.303847629806290e+00,
 5.212740720761116e+00,
-1.697806802249656e+00,
 3.603723294285617e-01,
-4.873193405327581e-02,
 3.115239986192253e-03,
-7.052248870239583e-05,
 5.975861393289898e-13,
-2.244300433086851e+01,
 6.381138411559539e+01,
-1.296062641440044e+01,
-1.732261722889801e+02,
 3.201844275030160e+02,
-2.877486645238489e+02,
 1.553970808919718e+02,
-5.263632804841900e+01,
 1.052424851243618e+01,
-8.052229681859826e-01,
-1.294834168322548e-01,
 3.491001117314470e-02,
-2.621648845901227e-03,
 4.369675047097478e-05,
 1.084703680918890e-06,
```

```
     2.614068553557722e-11
         };

/* Denominator coefficients for the controller */

static double den1[16]=
        {  -4.134112763067415e+00,
     7.952036535343285e+00,
    -9.679222466663568e+00,
     8.264755004139751e+00,
    -5.156472050247617e+00,
     2.466202573916116e+00,
    -9.526651742543444e-01,
     3.043804172191035e-01,
    -7.809450273438016e-02,
     1.507657409431068e-02,
    -2.062878392003597e-03,
     1.878287444794598e-04,
    -9.267211620365389e-06,
     1.776861871651543e-07,
    -4.692902509427273e-11,
    -3.103818469371239e-28
                };

/* Numerator coefficients for the plant */


static double num2[2][2][14]=
      {  -3.585263080640999e-05,
    -2.777807450549119e-05,
     4.539524991091781e-04,
    -5.958420322347280e-04,
     2.664995123886627e-04,
    -5.610107617126658e-06,
    -4.188469924792937e-05,
     1.857095302063017e-05,
    -3.454000472385976e-06,
     2.192568109666768e-07,
     3.819556833218761e-09,
    -4.268854429590967e-10,
    -2.655110714860240e-13,
    -5.118954168088873e-14,
    -7.513347186476338e-04,
     8.561991155699999e-03,
    -1.277978855932282e-02,
     1.181476279265681e-03,
     8.878987005693340e-03,
    -7.486010702827839e-03,
     2.988938381587536e-03,
    -6.418478904496455e-04,
     5.742794875096026e-05,
     1.381940029905864e-06,
    -4.971836625593487e-07,
     1.801323508010827e-08,
    -2.814170373926085e-11,
     2.083960044277704e-12,
    -4.056931727527413e-04,
     8.640985801235956e-04,
    -9.645374778415317e-04,
     7.668732134167300e-04,
```

```
    -3.355203324524325e-04,
     2.002988600113831e-05,
     4.423899844696599e-05,
    -2.134656278755809e-05,
     4.206573556321883e-06,
    -2.414898471374619e-07,
     3.935541586379038e-09,
    -1.296789423146734e-09,
     6.012915966444080e-11,
     6.139928164096763e-13,
    -1.468836867932399e-02,
     5.067239044670480e-02,
    -8.185463286921646e-02,
     8.351205855023736e-02,
    -5.856919312782694e-02,
     2.927929467703660e-02,
    -1.082648765944083e-02,
     2.950804399305770e-03,
    -5.601566387111508e-04,
     6.631958214738630e-05,
    -2.050337748895240e-06,
    -1.175920855414365e-07,
     4.286644733642849e-09,
     6.953985652098497e-11
        };


/* Denominator coefficients for the plant */

static double den2[14]=
        { -4.729739487034623e+00,
     9.756583616420810e+00,
    -1.169645031172698e+01,
     9.179964647228649e+00,
    -5.025697420203761e+00,
     1.978913683674686e+00,
    -5.630838727994958e-01,
     1.137480768724702e-01,
    -1.554252095215423e-02,
     1.300296942834350e-03,
    -5.737414053144769e-05,
     1.055438679886130e-06,
    -4.924500562373306e-09,
     1.114762000808644e-10
           };
```

```
/*****************************************************************************
 *  28 july 1991                                                             *
 *  transfer1614lim.txt       LIMITED CASE                                   *
 *  Header file for transfer function variables                              *
 *  Sampling time 0.02 sec                                                   *
 *  CAPT D. Bertrand                                                         *
 *                                                                           *
*****************************************************************************/

/* Define the case # */

#define   c        7      /* controller and plant in series (limited case) */


/* Define the variables */

/* Level 1 */
#define ord1       16      /* order of the controller */
#define num_in      2      /* number of inputs to the controller */
#define reg_vec1   64      /* regression vector =ord1*(num_in+num_out) */
#define num_feed   62      /* feedback layer =reg_vec-num_in */
#define num_comd   64      /* command layer = reg_vec */
#define num_cont   64      /* control layer = reg_vec */
#define num_out     2      /* output layer = num_out */

/* Level 2 */
#define ord2       14      /* order of the plant */
#define num_in2     2      /* number of inputs to the plant */
#define reg_vec2   56      /* regression vector =ord2*(num_in+num_out) */
#define num_feed2  54      /* feedback2 layer = reg_vec-num_out */
#define num_comd2  56      /* command2 layer= reg_vec */
#define num_cont2  56      /* control2 layer = reg_vec */
#define num_out2    2      /* output2 layer = num_in */


/* Declarations */

static double ts={0.02};
static double alpha1={1.0};
static double alpha2={1.0};
static double alpha3={0.001};

static char *input_name[]={"Illegal Input",
   "A Pulse input of 1 degree for 1 sec (input 1)","(input 2)"};

static char *state_name[]={"Illegal State","alpha(t)",
        "q(t)"};



/* Numerator coefficients for the controller */


static double num1[2][2][16]=
      {  -2.479711140348080e-01,
     1.388546204672132e+00,
    -3.546772405083324e+00,
```

```
 5.565578428016757e+00,
-6.095882090095444e+00,
 4.985447353467308e+00,
-3.146599111301740e+00,
 1.554675660772483e+00,
-6.059910645955577e-01,
 1.864047652459634e-01,
-4.437074509219546e-02,
 7.795354757026977e-03,
-9.144571099421236e-04,
 5.442429543013751e-05,
-1.205376035060677e-06,
 4.026048729360174e-10,
-2.613658394640916e-01,
 1.180069353917421e+00,
-2.158547648322319e+00,
 1.997245892440489e+00,
-8.979474879152747e-01,
 7.149916375388088e-02,
 1.116609373279331e-01,
-5.083590416542716e-02,
 8.459759671103129e-03,
-2.947817097250782e-04,
 1.181801294794504e-04,
-5.270969302269626e-05,
-1.464615832096148e-05,
 6.312847660653519e-06,
-6.006135234343973e-07,
 1.681353315103645e-08,
 2.536536100527425e-01,
-1.564660964790082e+00,
 4.360350547653734e+00,
-7.303031625580367e+00,
 8.252324911573620e+00,
-6.695221168985064e+00,
 4.052659667284534e+00,
-1.877182667414060e+00,
 6.754625922866597e-01,
-1.891390637720003e-01,
 4.050225911885580e-02,
-6.338009693289473e-03,
 6.545692523768167e-04,
-3.537724466211512e-05,
 7.202327264979368e-07,
 4.830176852017413e-10,
-3.483309500992426e-01,
 1.783428912751852e+00,
-3.819546231203830e+00,
 4.333227159290175e+00,
-2.566270736673481e+00,
 3.512741534094985e-01,
 6.441742008168490e-01,
-5.713335098825008e-01,
 2.503842147433657e-01,
-6.674350820441199e-02,
 1.043571874877550e-02,
-6.446155271970164e-04,
-6.960491892187438e-05,
 1.575972011659112e-05,
-9.835640847950234e-07,
```

```
     2.017172272019888e-08
               };

/* Denominator coefficients for the controller */

static double den1[16]=
        {   -6.625803214724185e+00,
     1.987480946391011e+01,
    -3.598199208859263e+01,
     4.432944619799812e+01,
    -3.979919039256840e+01,
     2.722951670474420e+01,
    -1.461991140558815e+01,
     6.267921584738075e+00,
    -2.160255681723939e+00,
     5.947802317875547e-01,
    -1.274873020013756e-01,
     2.017521031941172e-02,
    -2.124031340776859e-03,
     1.171829283955842e-04,
    -2.459886242548657e-06,
     1.353889590168705e-22,
        };


/* Numerator coefficients for the plant */


static double num2[2][2][14]=
      {   -3.585263080640999e-05,
    -2.777807450549119e-05,
     4.539524991091781e-04,
    -5.958420322347280e-04,
     2.664995123886627e-04,
    -5.610107617126658e-06,
    -4.188469924792937e-05,
     1.857095302063017e-05,
    -3.454000472385976e-06,
     2.192568109666768e-07,
     3.819556833218761e-09,
    -4.268854429590967e-10,
    -2.655110714860240e-13,
    -5.118954168088873e-14,
    -7.513347186476338e-04,
     8.561991155699999e-03,
    -1.277978855932282e-02,
     1.181476279265681e-03,
     8.878987005693340e-03,
    -7.486010702827839e-03,
     2.988938381587536e-03,
    -6.418478904496455e-04,
     5.742794875096026e-05,
     1.381940029905864e-06,
    -4.971836625593487e-07,
     1.801323508010827e-08,
    -2.814170373926085e-11,
     2.083960044277704e-12,
    -4.056931727527413e-04,
     8.640985801235956e-04,
    -9.645374778415317e-04,
```

```
     7.668732134167300e-04,
    -3.355203324524325e-04,
     2.002988600113831e-05,
     4.423899844696599e-05,
    -2.134656278755809e-05,
     4.206573556321883e-06,
    -2.414898471374619e-07,
     3.935541586379038e-09,
    -1.296789423146734e-09,
     6.012915966444080e-11,
     6.139928164096763e-13,
    -1.468836867932399e-02,
     5.067239044670480e-02,
    -8.185463286921646e-02,
     8.351205855023736e-02,
    -5.856919312782694e-02,
     2.927929467703660e-02,
    -1.082648765944083e-02,
     2.950804399305770e-03,
    -5.601566387111508e-04,
     6.631958214738630e-05,
    -2.050337748895240e-06,
    -1.175920855414365e-07,
     4.286644733642849e-09,
     6.953985652098497e-11
               };

/* Denominator coefficients for the plant */

static double den2[14]=
        { -4.729739487034623e+00,
     9.756583616420810e+00,
    -1.169645031172698e+01,
     9.179964647228649e+00,
    -5.025697420203761e+00,
     1.978913683674686e+00,
    -5.630838727994958e-01,
     1.137480768724702e-01,
    -1.554252095215423e-02,
     1.300296942834350e-03,
    -5.737414053144769e-05,
     1.055438679886130e-06,
    -4.924500562373306e-09,
     1.114762000808644e-10
            };
```

APPENDIX D: MATLAB M-FILES

```
% contdisclim1.m
% input files:...hinflim.mat
% output files:...tflim1/2.mat, contdisclim1/2/a/q.met, truelim1/2/a/q.mat


%"THIS M-FILE CREATES  DISCRETE & CONTINUOUS BODE PLOTS FOR THE PLANT
% STATE REPRESENTATION OF Hinf LIMITED PERFORMANCE X-29 model, CASE #2.
% Altitude = 30000 feet
% Mach# = .5


% The A,B,C and D matrices are first balanced to measure a reasonable condition
% number for the A matrix, then converted from a continuous to a discrete state
% space model using Ts=0.02 sec as the sampling time, and finally converted to
% a transfer function form using,
%
%                              H(z)= C*inv(zI-A)*B= Y(z)/U(z)                (1)
%
% By replacing the z-transform with the 1/q backward shift operator, the
% numerator and denominator terms of that transformation may be used to obtain
% the DARMA model,
%
%                              A(q)y(t) = B(q)u(t)                           (2)
%
% Reworking equation (2) gives [Ref. 7:pp. 71-72],
%
%                               y(t) = B(q)u(t) - (A(q) - 1)y(t)             (3)
%
% Expanding the matrix polynomials and rearranging it to obtain two recursive
% equations of the form similar to equation (3.8) of Chapter III gives,
%
%                               Yi(t) = SUMj[Bij*Uj(t-j) - SUMj[Aj*Yj(t-j)]  (4)
%
%      where SUM indicates a summation operation
%            i   indicates the number of outputs
%            j   indicates the number of past input and output measurements
%            Aj  terms are the denominator coefficients
%            Bij terms are the numerator coefficients
%
% The algorithm referring to the control input in USERIO of the case request
% RQ_LEARNRSLT of Appendix A, is represented by equation (4). Control [j]
% specifies the first 30 elements of the control layer, which are the past 30
% inputs, [Uj(t-j)], whereas control [30*(i+1)+j] specifies the second and the
% third block of 30 elements, which are the past 30 outputs, [Yj(t-j)], for
% each of the two ouput elements.
%        Finally, the coefficients for B(q) and A(q)-1 matrices of equation (3)
% are represented in the Transfer.h files as the numerator and denominator
% coefficients for both cases, Optimal and Limited performance.


% get the a,b,c & d matrices

load hinflim.mat

a=acgf;
b=bcgf;
```

```
c=ccgf;
d=dcgf;

%  balance a ,b and c matrices.

[ab,bb,cb,g,t]=obalreal(a,b,c);


% compute continuous Bode

disp(' calculating continuous Bode  please wait')

w=logspace(-3,2,1024);
[magc,phasec]=bode(ab,bb,cb,d,1,w);
mag=20*log10(mag);


% Compute discrete time system

t=.02;      % SAMPLING TIME

[ad,bd]=c2d(ab,bb,t);


% Compute transfer function for the two inputs

[num1,den1]=ss2tf(ad,bd,cb,d,1);
[num2,den2]=ss2tf(ad,bd,cb,d,2);


%save tflim1 num1 den1
%save tflim2 num2 den2


% Discrete Bode plot calculations

disp('')
disp(' calculating discrete Bode Nyquist 50Hz   T=.02 ')


[mag,phase]=dbode(ad,bd,cb,d,1,w);

loglog(w,magc(:,1),50*w,mag(:,1))
title('X-29 Continuous and Discrete Alpha Frequency Response')
title('                Optimal case - input 1                ')
xlabel('Altitude = 30000 feet    frequency (Hz)     Mach =.5'),
ylabel('magnitude ')
text(0.01,.0001,'continuous ___')
text(.01,.00001,'discrete ---')
meta contdisclim1a
pause


loglog(w,magc(:,2),50*w,mag(:,2))
title('X-29 Continuous and Discrete Q  Frequency Response ')
title('                Optimal case - input 2                ')
xlabel('Altitude = 30000 feet    frequency (Hz)     Mach =.5'),
ylabel('magnitude ')
text(0.1,.001,'continuous ___')
text(.1,.0003,'discrete ---')
```

```
meta contdisclim1q

f=w;
m=mag(:,1);
%save truelim1a f m

m=mag(:,2);
%save truelim1q f m
```

```
% wgtm.m

% This matlab file calculates the SVD of the weight matrices,
% which are composed of the connections weights between the
% two hidden layers. The first hidden layer has 42 elements
% and the second hidden layer has been tested with 30, 21,
% 12, 8 and 5 elements.

% The input .nnp files are from the optimal controller network 1
% of case #6 or case6h2.nnd.


load c6h230.nnp                 % Load the .nnp files.
load c6h221.nnp
load c6h212.nnp
load c6h28.nnp
load c6h251.nnp
load c6h22.nnp

[m,n1]=size(c6h230);            % Obtain the size of each matrix.
[m,n2]=size(c6h221);
[m,n3]=size(c6h212);
[m,n4]=size(c6h28);
[m,n5]=size(c6h251);
[m,n6]=size(c6h22);

tf1=c6h230(:,3:n1);             % Get rid-off the first two elements.
tf2=c6h221(:,3:n2);
tf3=c6h212(:,3:n3);
tf4=c6h28(:,3:n4);
tf5=c6h251(:,3:n5);
tf6=c6h22(:,3:n6);

t=0:100:9900;                   % Time vector.

a1=zeros((n1-2)/30,30);         % Divide the matrix into blocks of which
a2=zeros((n2-2)/21,21);         % the number of columns equals the number
a3=zeros((n3-2)/12,12);         % of elements in the second hidden layer.
a4=zeros((n4-2)/8,8);
a5=zeros((n5-2)/5,5);
a6=zeros((n6-2)/2,2);

for i=1:m                       % Calculate the actual SVD's.
    a1(:)=tf1(i,:);
    s1(:,i)=svd(a1);
    a2(:)=tf2(i,:);
    s2(:,i)=svd(a2);
    a3(:)=tf3(i,:);
    s3(:,i)=svd(a3);
    a4(:)=tf4(i,:);
    s4(:,i)=svd(a4);
    a5(:)=tf5(i,:);
    s5(:,i)=svd(a5);
    a6(:)=tf6(i,:);
    s6(:,i)=svd(a6);
end;

i=find(s1<.6);                  % Get rid-off the insignificant values.
s1(i)=zeros(i);
i=find(s2<.54);
```

```
s2(i)=zeros(i);
i=find(s3<.45);
s3(i)=zeros(i);
i=find(s4<.39);
s4(i)=zeros(i);
i=find(s5<.35);
s5(i)=zeros(i);
i=find(s6<.30);
s6(i)=zeros(i);

axis([0 10000 0.1 6])

plot(t,s1',’-’)
title('SVD plot of the wgt matrix (hidden 2 - 30 elements)')
xlabel('Number of Epochs')
ylabel('SVD')
meta svd41
pause

plot(t,s2',’-’)
title('SVD plot of the wgt matrix (hidden 2 - 21 elements)')
xlabel('Number of Epochs')
ylabel('SVD')
meta svd42
pause

plot(t,s3',’-’)
title('SVD plot of the wgt matrix (hidden 2 - 12 elements)')
xlabel('Number of Epochs')
ylabel('SVD')
meta svd43
pause

plot(t,s5',’-’)
title('SVD plot of the wgt matrix (hidden 2 - 5 elements)')
xlabel('Number of Epochs')
ylabel('SVD')
meta svd44
pause


plot(t,s6',’-’)
title('SVD plot of the wgt matrix (hidden 2 - 2 elements)')
xlabel('Number of Epochs')
ylabel('SVD')
meta svd45
pause
```

```
% spcallim.m
% input files:...inputlim1/2.nnp, plantlim1/2.nnp
% output files:...speclim1/2.mat

% THIS M-FILE CREATES THE SPECTRUM RETURNS OF CASE #2 OF BOTH INPUTS
% WITH A 5 ARRAYS FUNCTION P=[Pxx,Pyy,Pxy,Txy,Cxy] WHERE Txy IS THE
% COMPLEX TRANSFER FUNCTION FROM X TO Y. P=SPECTRUM(X,Y,M) WHERE X= INPUT
% VECTOR, Y=OUTPUT VECTOR, AND M=2048 PTS, WHICH DIVIDES BOTH VECTORS INTO
% SECTIONS OF 2048 POINTS EACH.
% PERFORM SPECTRAL ANALYSES ON THE TWO SEQUENCES X AND Y.


load inputlim1.nnp
load plantlim1.nnp
load inputlim2.nnp
load plantlim2.nnp

x1=inputlim1(1:2049,3);
y1=plantlim1(1:2049,3);
z1=plantlim1(1:2049,4);
x2=inputlim2(1:2049,3);
y2=plantlim2(1:2049,3);
z2=plantlim2(1:2049,4);

clear inputlim1 plantlim1 inputlim2 plantlim2

disp(1)
Pa1=spectrum(x1,y1,2048,1024);

disp(2)
Pq1=spectrum(x1,z1,2048,1024);

disp(3)
Pa2=spectrum(x2,y2,2048,1024);

disp(4)
Pq2=spectrum(x2,z2,2048,1024);

save speclim1 Pa1 Pq1;
save speclim2 Pa2 Pq2;
```

```
% splotslim1.m

% input files: ...speclim1/2.mat, truelim1/2/a/q.mat
% output files:... TFlim1/2/a/q.met

% THIS M-FILE COMPARES THE SYSTEM AND NETWORK ALPHA AND Q "FREQUENCY" RESPONSES
% FOR CASE #2 (LIMITED CASE) WITH RESPECT TO THE TWO INPUTS.


load speclim1;
load speclim2;

[n,m] = size(Pa1);
Fs=100;
f1 = (1:n-1)/n*Fs/2;



load truelim1a;
loglog(pi*f1,abs(Pa1(2:n,4)),50*f,m)
title('Txy - Transfer function for Alpha (Limited case -input 1)')
xlabel('  Altitude= 30,000 feet     frequency(hz)    Mach= 0.5  ')
ylabel('Magnitude')
text(1.0,.001,'True system ---')
text(1.0,.0005,'40k cycles ___')
meta TFlima1
pause


load truelim1q;
loglog(pi*f1,abs(Pq1(2:n,4)),50*f,m), ...
title('Txy - Transfer function for Q (Limited case -input 1)')
xlabel('  Altitude= 30,000 feet     frequency(hz)    Mach= 0.5  ')
ylabel('Magnitude')
text(0.3,.02,'True system ---')
text(0.3,.008,'40k cycles ___')
meta TFlimq1
pause

load truelim2a;
loglog(pi*f1,abs(Pa2(2:n,4)),50*f,m)
title('Txy - Transfer function for Alpha (Limited case -input 2)')
xlabel('  Altitude= 30,000 feet     frequency(hz)    Mach= 0.5  ')
ylabel('Magnitude')
text(1.0,.001,'True system ---')
text(1.0,.0005,'40k cycles ___')
meta TFlima2
pause


load truelim2q;
loglog(pi*f1,abs(Pq2(2:n,4)),50*f,m), ...
title('Txy - Transfer function for Q (Limited case -input 2)')
xlabel('  Altitude= 30,000 feet     frequency(hz)    Mach= 0.5  ')
ylabel('Magnitude')
text(0.3,.02,'True system ---')
text(0.3,.008,'40k cycles ___')
meta TFlimq2
pause
```

```
% lim1.m
% input files:...hinflim.mat, steplim1/2.nnp
% ouput files:...lim1/2/a/q.met

% THIS M-FILE COMPARES THE SYSTEM AND NETWORK ALPHA & Q "TIME" RESPONSE FOR
% CASE #2 (LIMITED CASE) WITH RESPECT TO THE TWO INPUTS.

load hinflim.mat
load steplim1.nnp
load steplim2.nnp
format long e

w1=steplim1(1:251,3);
z1=steplim1(1:251,4);
w2=steplim2(1:251,3);
z2=steplim2(1:251,4);

clear steplim1 steplim2

time1=[0:0.02:5];    % for the network response


% u1- input 1 and u2- input 2.

u1=[ones(1,101) zeros(1,300);zeros(1,401)]';
u2=[zeros(1,401);ones(1,101) zeros(1,300)]';

[y1]=lsim(acgf,bcgf,ccgf,dcgf,u1,time1);
[y2]=lsim(acgf,bcgf,ccgf,dcgf,u2,time1);


plot(time1,y1(:,1),time1,z1)
title('X-29  DESIRED AND ACTUAL ALPHA  RESPONSE (limited case -input 1)')
xlabel('TIME - SEC')
ylabel('DEGREES')
text(1.75,.6,'Desired ---')
text(.5,.12,'Actual ---')
grid
%meta lim1a
pause

plot(time1,y1(:,2),time1,w1)
title('X-29  DESIRED AND ACTUAL Q RESPONSE(limited case -input 1)')
xlabel('TIME - SEC')
ylabel('DEGREES')
text(1.75,.6,'Desired ---')
text(.5,.12,'Actual ---')
grid
%meta lim1q
pause

plot(time1,y2(:,1),time1,z2)
title('X-29  DESIRED AND ACTUAL ALPHA  RESPONSE (limited case -input 2)')
xlabel('TIME - SEC')
ylabel('DEGREES')
text(1.75,.6,'Desired ---')
text(.5,.12,'Actual ---')
grid
%meta lim2a
pause
```

```
plot(time1,y2(:,2),time1,w2)
title('X-29  DESIRED AND ACTUAL Q RESPONSE(limited case -input 2)')
xlabel('TIME - SEC')
ylabel('DEGREES')
text(1.75,.6,'Desired ---')
text(.5,.12,'Actual ---')
grid
%meta lim2q
pause
```

APPENDIX E: TABLE OF CONFIGURATIONS AND CASES

Configuration	case #	Model Structure Level 1	Level 2	Control Strategy	Header File
Simulation of the X-29 Closed-Loop Plant	1	MIMO (Optimal)	n/a	contstr1	Transfer30hp
	2	MIMO (Limited)	n/a	contstr1	Transfer30lim
Identif. of the Inverse Plant	3	SIMO (case #1)	* MISO (inv. case #1)	contstr1 &2	Transferhp1inv
	4	SIMO (A4 Plant)	** MISO (A4inv.Plant)	contstr1 & 2	TransferA4inv
	5	SIMO (x-29 Plant)	*** MISO (x29Inv.Plan)	contstr1 & 2	Transfer14inv
Simulation of the Existing Controllers and the Plant	6	MIMO (X-29cont.) (Optimal)	MIMO (X-29 plant) (Optimal)	contstr2	transfer1614hp
	7	MIMO (X-29cont.) (Limited)	MIMO (X-29 plant) (Limited)	contstr2	transfer1614lim
	8	**** Part I: closure of the open-loop model of case #6 Part II: closure of the open-loop model of case #7			

* 30th order X-29 transfer function whose inverse is stable.
** 4th order A-4D plant whose inverse is unstable.
*** 14th order X-29 plant whose inverse is very unstable.
**** The closure implies the connection in series of the controller and the plant, and the intrusion of the negative feedback loop of gain 1 from the plant outputs to the controller inputs.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, California 93943-5000	2
3.	Chairman, Code 47 Department of Aeronautics and Astronautics Naval Postgraduate School Monterey, California 93943-5000	1
4.	Professor D. J. Collins Code 67Co Department of Aeronautics and Astronautics Naval Postgraduate School Monterey, California 93943-5000	3
5.	Professor R. Cristi EC/CX Department of Electrical and Computer Eng. Naval Postgraduate School Monterey, California 93943-5000	1
6.	Maj. Al Smigelski Directorate of Avionics, Simulations and Photography DASP 2-3 NDHQ MGen George R. Pearkes Building Ottawa, Canada K1A 0K2	1
7.	Maj. J.D Bernier DLAEEM 4-2 NDHQ MGen George R. Pearkes Building Ottawa, Canada K1A 0K2	1
8.	Lt. C.N Gedo U.S. Naval Ship Repair Facility Box 34 FPO San-Fransisco, 96651-1400	1

9. Capt. J.D. Bertrand
Project Management Office
Anti-Armored Project
NDHQ MGen George R. Pearkes Building
Ottawa, Canada K1A 0K2

Thesis
B45819 Bertrand
c.1 Application of neural network to adaptive control theory for super-augmented aircraft.